VMS Software

VSI OpenVMS x86-64 V9.2-2 Release Notes

VSI OpenVMS x86-64 V9.2-2 Release Notes

VMS Software

Preface

1. Introduction

VMS Software, Inc. (VSI) is pleased to introduce VSI OpenVMS x86-64 V9.2-2.

2. Intended Audience

This document is intended for all users of VSI OpenVMS x86-64 V9.2-2. Read this document before you install or use VSI OpenVMS x86-64 V9.2-2.

3. Related Documents

The following documents provide additional information in support of this release:

*   *VSI OpenVMS x86-64 V9.2-2 Installation Guide [https://docs.vmssoftware.com/vsi-openvms-x86-64-v922-installation-guide]*

*   *VSI OpenVMS x86-64 Boot Manager User Guide [https://docs.vmssoftware.com/vsi-openvms-x86-64-boot-manager-user-guide-922/]*

*   *VSI OpenVMS Calling Standard Manual [https://docs.vmssoftware.com/vsi-openvms-calling-standard/]*

*   *VSI OpenVMS Linker Utility Manual [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/]*

*   *VSI OpenVMS x86-64 Cross-Tools Kit Installation and Startup Guide [https://docs.vmssoftware.com/vsi-x86-64-cross-tools-kit-installation-and-startup-guide-v922/]*

Chapter 1. Read These Before You Start

Before you download the VSI OpenVMS x86-64 V9.2-2 installation kit, VSI strongly recommends that you read the notes in this section. These notes provide information about the hypervisors tested by VSI, CPU feature checks for virtual machines, terminal emulator settings, and licensing on OpenVMS x86-64 systems.

Note that if an entry describing a problem or a change in one of the previous release notes (especially for field test versions) is not present in the current release notes, then it is either no longer applicable or has been fixed.

1.1. Supported Disk Types

VSI OpenVMS x86-64 V9.2-2 supports SATA disks across all supported hypervisors.

On Oracle VirtualBox, VSI OpenVMS also supports VirtIO-SCSI disks.

On VMware ESXi, VSI OpenVMS also supports the configuration of up to four (4) LSI Logic Parallel SCSI controllers per virtual machine. LSI Logic SAS controllers are not supported.

On KVM, VSI OpenVMS also supports SCSI and VirtIO-SCSI disks.

1.1.1. VSI OpenVMS Does Not Support Thin-Provisioned Volumes on Any Architecture

VSI OpenVMS *does not* support thin-provisioned volumes on any architecture. Attempting to use thin-provisioned virtual disks may result in loss or corruption of data on the device. VSI currently has no plans to add thin provisioning support on any architecture.

When creating new virtual volumes for VSI OpenVMS, *do not* enable thin provisioning, even if the hypervisor that you are using supports it.

Note that HPE OpenVMS versions *do* support thin provisioning. If you are currently using a thin-provisioned disk on your HPE OpenVMS system and you are planning to upgrade to VSI OpenVMS, you must change the provisioning of your virtual disk to thick provisioning.

For information on changing the provisioning of a virtual disk, refer to VMware documentation.

If you have any further questions, please contact VSI support via `<support@vmssoftware.com>`.

1.2. Tested Platforms

VSI OpenVMS x86-64 V9.2-2 can be installed as a guest operating system on Oracle VM VirtualBox, KVM, and VMware virtual machines using the **X860922OE.ISO** file.

VirtualBox

VSI tests with VirtualBox V7.0.x and regularly installs patches when they are available.

KVM

For KVM, VSI recommends ensuring that your system is up-to-date with KVM kernel modules and the associated packages necessary for your particular Linux distribution.

VSI has tested VSI OpenVMS x86-64 V9.2-2 with KVM on several Linux distributions. The following table includes the Linux distribution, version, and the QEMU version:

Linux Distribution	QEMU Version (package information)
openSUSE Leap 15.4	6.2.0 (SUSE Linux Enterprise 15)
Rocky Linux 8.6	6.2.0 (qemu-kvm-6.2.0-11.module +el8.6.0+1052+ff61d164.6)
Rocky Linux 8.7	6.2.0 (qemu-kvm-6.2.0-32.module +el8.8.0+1279+230c2115)
Rocky Linux 9.1	7.0.0-13
Ubuntu 22.04 LTS	6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.15)

VMware

VSI has tested VSI OpenVMS x86-64 V9.2-2 with the following VMware products:

VMware Product	Version Tested by VSI
Workstation Pro	V16.2.x, V17.0.x
Workstation Player	V17.0.x
Fusion Pro	V13.0.x
Fusion Player	V12.2.x
ESXi	V6.7.x, V7.0.x, V8.0.x (with compatibility of V8, V7U2, V7U1, V6.7U2)
vSphere Client	V8.0

Warning

If you choose to upgrade from a previous version of **VMware Fusion** to VMware Fusion **13**, from a previous version of **VMware Workstation** to VMware Workstation **17**, or from a previous version of **VMware ESXi** to VMware ESXi **8**, you *will not* be able to run any VMs with VSI OpenVMS x86-64 versions prior to V9.2-1. However, VMs running VSI OpenVMS x86-64 V9.2 *can* be upgraded to VSI OpenVMS x86-64 V9.2-1 or later which *will* run under VMware Fusion 13/VMware Workstation 17/ VMware ESXi 8.

VMware Licenses

Note that *not* all VMware license types are currently supported for running VSI OpenVMS x86-64 V9.2-2. The following table lists VMware license types that have been tested by VSI.

VMware License	VSI Tested
Enterprise Plus	ESXi V6.7.x, V7.0.x, and V8.0.x

VMware License	VSI Tested
Enterprise	Not tested
Essentials Plus	Not currently supported
Essentials	Not currently supported
Standard	Not currently supported
Hypervisor	Not currently supported

The VMware licenses that are marked as "Not currently supported" do not support the use of virtual serial lines in a virtual machine. OpenVMS requires a serial port connection with a terminal emulator, and therefore VMware systems with these licenses are not currently supported for running OpenVMS. This support will be added in a future release of VSI OpenVMS x86-64.

1.3. MD5 Checksum for the X860922OE.ZIP File

VSI recommends that you verify the MD5 checksum of the **X860922OE.ZIP** file after it has been downloaded from the VMS Software Service Platform to your target system. The MD5 checksum of **X860922OE.ZIP** must correspond to the following value:

BBE9631E781B0E8CC764C5EAA61CE2A2

To calculate the MD5 checksum, you can use any platform-specific utilities or tools that are available for your system.

1.4. CPU Compatibility Checks for Virtual Machines

VSI OpenVMS x86-64 requires that the CPU supports certain features that are not present in all x86-64 processors. When using virtual machines, both the host system and guest virtual machine must have the required features.

Host System Check

Before downloading the VSI OpenVMS x86-64 V9.2-2 installation kit, VSI recommends that you determine whether your host system has the required CPU features to run VSI OpenVMS x86-64. For this purpose, execute a Python script called **vmscheck.py** on your host system. This script, along with the accompanying PDF document entitled *VMS CPUID Feature Check*, can be downloaded from the official VSI OpenVMS x86-64 web page [https://vmssoftware.com/openkits/alpopensource/vmscheck.zip].

The *VMS CPUID Feature Check* document contains instructions on how to run the script and interpret the results and also describes script limitations.

Guest Virtual Machine Check

The OpenVMS Boot Manager performs the CPU feature check on the guest virtual machine. The CPU feature check is performed automatically every time the Boot Manager is launched. If the check has passed successfully, the following message is displayed:

```
Checking Required Processor Features:      PASSED
```

In addition, before booting VSI OpenVMS x86-64 V9.2-2, you can issue the following Boot Manager command to list the compatibility details:

```
BOOTMGR> DEVICE CPU
```

Important

VSI OpenVMS x86-64 will not boot on the system that fails either of the following CPU feature checks:

- The host system check (via the **vmscheck.py** script)

- The guest virtual machine check (via the OpenVMS Boot Manager).

Note

In case the system has the required CPU features but lacks some of the optional CPU features, the OpenVMS operating system may have noticeably lower performance.

1.5. Terminal Emulator Settings

When you are in the Boot Manager and before proceeding with the installation of VSI OpenVMS x86-64, you are required to access the system through a serial port connection with a terminal emulator, such as PuTTY. You may need to experiment in order to find the appropriate setting for your emulator.

Refer to the _VSI OpenVMS x86-64 Boot Manager User Guide_ [https://docs.vmssoftware.com/vsi-openvms-x86-64-boot-manager-user-guide-922] and _VSI OpenVMS x86-64 V9.2-2 Installation Guide_ [https://docs.vmssoftware.com/vsi-openvms-x86-64-v922-installation-guide/] for more details on the emulator settings.

On Windows, VSI recommends using PuTTY. Some PuTTY users have found success with the following settings:

- If the connection type is **Raw**, the following settings should be used:

Category	Setting	Value
Session	Connection type	Raw
Terminal	Implicit CR in every LF	Unchecked
Terminal	Implicit LF in every CR	Unchecked
Terminal	Local echo	Force off
Terminal	Local line editing	Force off (character mode)

- If the Connection type is **Telnet**, the following settings should be used:

Category	Setting	Value
Session	Connection type	Telnet

Category	Setting	Value
Connection > Telnet	Telnet negotiation mode	Switch from **Active** to **Passive**. This yields a connection to a PuTTY window.
Connection > Telnet	Telnet negotiation mode	Uncheck **Return key sends new line instead of ^M**

Note

As there is no Telnet server on the virtual machine host for the console communication, it is not literally a Telnet connection, but it can be used because not all emulators support a Raw connection.

1.6. Memory Disk and the Command Procedure SYS$MD.COM

VSI OpenVMS x86-64 uses a boot method called Memory Disk that simplifies the boot process by eliminating boot complexity and decoupling the VSI OpenVMS Boot Manager (chain loader) from a specific device or version of VSI OpenVMS x86-64. Memory Disk is created and deployed automatically during the installation process.

The Memory Disk contains all files that are required to boot the minimum OpenVMS kernel and all files needed to write system crash dumps. Changes such as file modifications or PCSI kit/patch installations require the operating system to execute a procedure to update the Memory Disk container, thus assuring that the next boot will use the new images. A command procedure, SYS$MD.COM, keeps the Memory Disk up-to-date.

Note

Do not invoke SYS$MD.COM directly unless you are advised to do so by VSI Support, or when required while following documented procedures. For example, if you load a user-written execlet by running SYS$UPDATE:VMS$SYSTEM_IMAGES.COM, you must then invoke SYS$UPDATE:SYS$MD.COM. For more details, see Section 3.3.8 of this document.

Note

VSI does not recommend changing or manipulating SYS$MD.DSK or SYS$EFI.SYS (or the underlying EFI partition) in any way. These files are needed to boot the system and are maintained by OpenVMS.

1.7. x86-64 Licensing

During the installation, you will be prompted to register the Product Authorization Keys (License PAKs) that you own.

A PAK is represented as a text structure containing a series of named fields and unique values that were generated by VSI. When prompted to register your PAKs, you have the option of deferring that until after the installation (which is recommended) and then registering them as a script (or using the LICENSE utility to enter values). If you choose to enter your PAKs during the installation, you can either

type the values of each requested field manually, or copy-and-paste the values into the console (assuming your console connection supports this action, which terminal emulators do).

Below is an example of a license text structure inside a PAK:

```
$ LICENSE REGISTER OPENVMS-X86-HAOE -
/ISSUER=VSI -
/AUTHORIZATION=1-VSI-20220608-00000 -
/PRODUCER=VSI -
/UNITS=1 -
/TERMINATION_DATE=31-OCT-2022 -
/OPTIONS=(PCL,X86_64) -
/CHECKSUM=2-XXXX-XXXX-XXXX-XXXX
```

1.7.1. Using a VMware vSphere Hypervisor Basic License

Use this procedure to run VMware vSphere Hypervisor on an ESXi server with a basic license (not Enterprise or Enterprise Plus). You must use serial ports within the same ESX server.
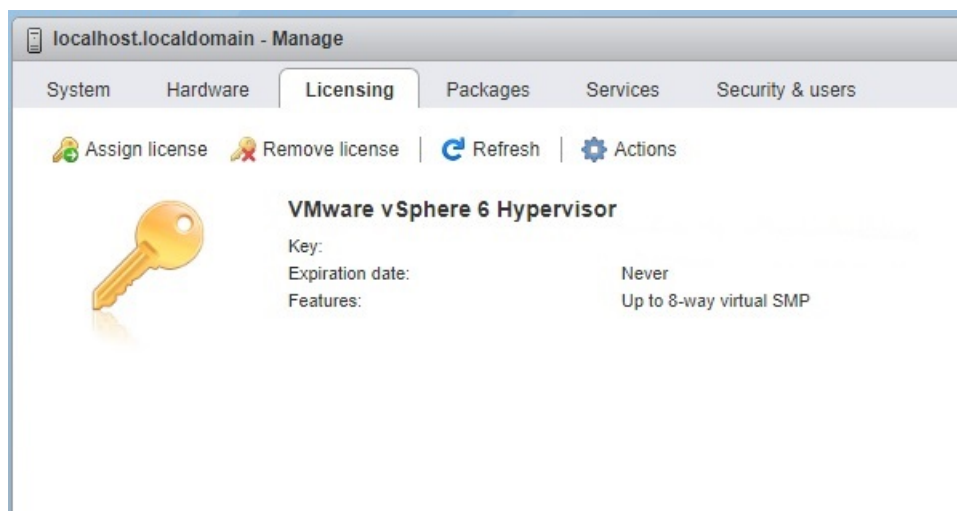
Using the named pipe functionality, map COM1/OPA0: on the VMS virtual machine to a pipe on a management server on which a terminal emulator is installed.

With the VM system in client mode, use the following syntax:

```
\\.\pipe_name
```

With the management system in server mode, use the following syntax:

```
\\.\pipe_name
```

where `pipe_name` is a unique string for each VM.

The terminal emulator should be set for serial connection at 115200 baud.

The two figures below show how to set up pipes for a local terminal-emulator-based console.
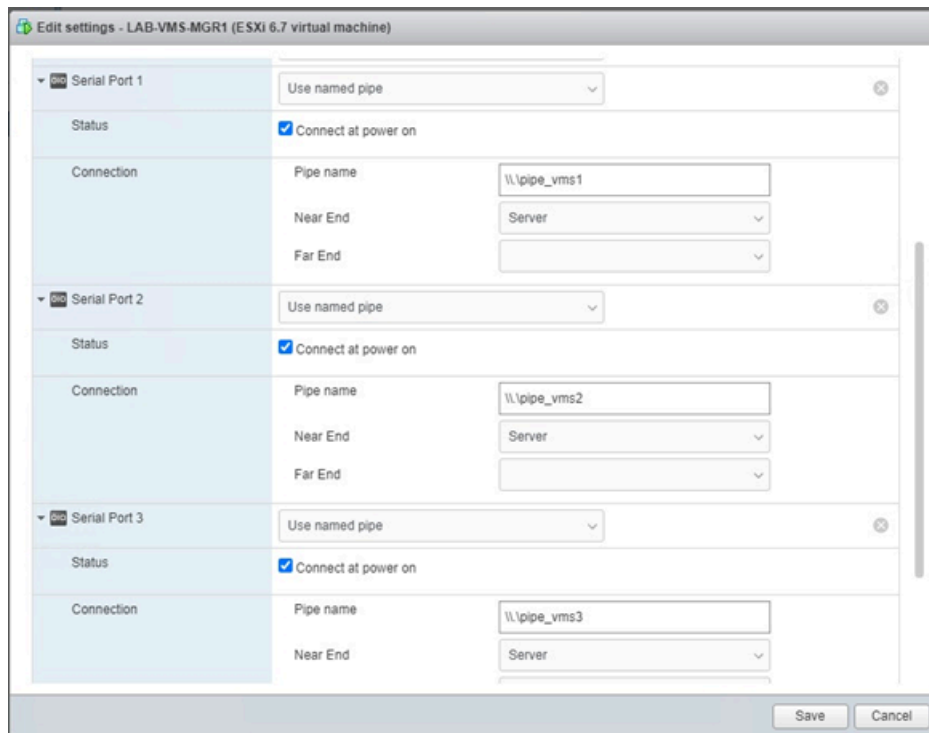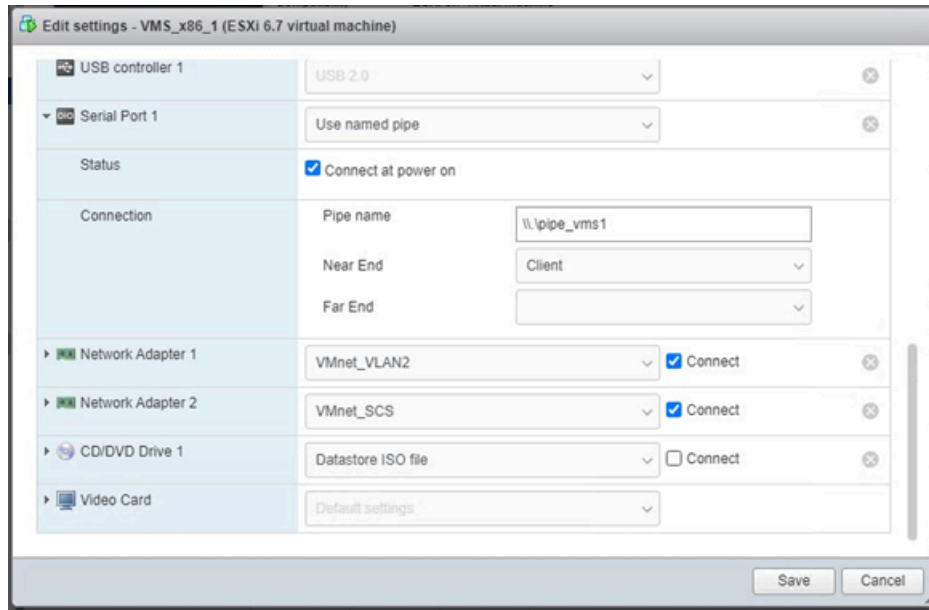
The two figures below show how to set up pipes between two local virtual machines where one plays the role of VMS console. This could be a virtual machine guest running any OS that supports a terminal emulator.

1.8. Networking Options

VSI OpenVMS x86-64 V9.2-x provides support for the following network stacks:

- VSI TCP/IP Services

- VSI DECnet Phase IV

- VSI DECnet-Plus

VSI TCP/IP Services V6.0-23 is part of the OpenVMS x86-64 V9.2-2 installation and will be installed along with the operating system. For more information, refer to *VSI TCP/IP Services Version V6.0 Release Notes* and other VSI TCP/IP Services documentation available at <u>docs.vmssoftware.com [https:// docs.vmssoftware.com/]</u>.

During the OpenVMS x86-64 V9.2-2 installation, you will also be offered the option to install either VSI DECnet Phase IV or VSI DECnet-Plus. Note that if you plan to use DECnet, you *must choose* between VSI DECnet Phase IV and VSI DECnet-Plus. *Only one* of these products can be installed on your system at a time.

1.8.1. VSI DECnet

Install either VSI DECnet Phase IV or VSI DECnet-Plus on VSI OpenVMS x86-64 V9.2-2 and then configure the product you have chosen, just as you would for an OpenVMS Alpha or OpenVMS Integrity release.

If you have DECnet Phase IV installed on your system and you want to use DECnet-Plus, you have to uninstall DECnet Phase IV and then install and configure DECnet-Plus.

Note

If your DECnet installation was *not* part of the main installation procedure for OpenVMS x86-64, you *must* update the Memory Disk after you install DECnet. The Memory Disk update ensures that SYS$NETWORK_SERVICES.EXE is loaded on boot. Use the following commands:

```
$ @SYS$UPDATE:SYS$MD.COM
```

After the next system reboot, you may want to purge the Memory Disk.

```
$ PURGE SYS$LOADABLE_IMAGES:SYS$MD.DSK
```

If you install DECnet as part of the main OpenVMS x86-64 installation procedure, you do *not* need to update the Memory Disk. The Memory Disk is updated at the end of the OpenVMS x86-64 installation.

After DECnet has been installed and configured, you can set host and copy files to/from other systems running DECnet.

1.8.2. Empty File for DECnet-Plus

The OpenVMS x86-64 installation procedure provides an empty file NET$CONFIG.DAT before installing the DECnet-Plus kit.

1.8.3. Bridged Networking

To understand how to configure your virtual machine network (devices and network configuration), please consult the VirtualBox, KVM, and VMware documentation. Some configurations may be incompatible with the operation of OpenVMS applications. For example, configuring a bridged adapter with a MacVTap device may inhibit the MAC address change done by DECnet, and configuring DECnet on VirtualBox may require allowing promiscuous mode on the NIC.

1.8.4. VSI FTP Service Might Not Connect Correctly In Virtual Environments

If the FTP service does *not* work after it has been started, switch to passive mode with the following command:

```
FTP> SET PASSIVE ON
Passive is ON
```

In passive mode, the FTP client always initiates a data connection. This is useful in virtual machine environments when there is network address translation (NAT) in your network.

To run this command automatically when you invoke FTP, put it into SYS$LOGIN:FTPINIT.INI. For the full description of the `SET PASSIVE` command, refer to the *VSI TCP/IP Services for OpenVMS User's Guide* [https://docs.vmssoftware.com/vsi-tcpip-services-for-openvms-user-s-guide-60/ #FTP_COMMANDS_SEC].

1.9. OpenSSL Update

VSI SSL3 V3.0-11 is the default SSL offering on VSI OpenVMS V9.2-2.

VSI SSL111 V1.1-1W is also available in this release in order to allow any existing SSL-based customer applications to continue to run. VSI SSL3 is designed to co-exist in parallel with VSI SSL111 by means of using different symbols for different versions.

Warning

SSL111 *will not* receive updates in future releases, so VSI strongly recommends that applications still using VSI SSL111 be moved to VSI SSL3.

1.10. VSI OpenSSH V8.9-1G for OpenVMS

VSI OpenSSH V8.9-1G is a required layered product that will be installed unconditionally with VSI OpenVMS x86-64 V9.2-2.

Warning

If you are upgrading to VSI OpenVMS V9.2-2 from an earlier version with OpenSSH V8.9-1F or earlier installed, make sure to uninstall VSI OpenSSH from your system *before* you start the upgrade procedure.

For post-installation and configuration instructions, refer to the *VSI OpenVMS x86-64 V9.2-2 Installation Guide* [https://docs.vmssoftware.com/vsi-openvms-x86-64-v922-installation-guide/#networkingChap].

For a detailed description of the features and bug fixes included in this release of OpenSSH V8.9, please refer to https://www.openssh.com/txt/release-8.9.

1.11. VSI Kerberos V3.3-2A for OpenVMS

VSI OpenVMS x86-64 V9.2-2 includes VSI Kerberos V3.3-2A for OpenVMS.

1.12. VSI DECwindows Motif V1.8 for OpenVMS

VSI OpenVMS x86-64 V9.2-2 includes VSI DECwindows Motif V1.8.

1.13. Required Layered Products

The following layered products will be installed unconditionally with VSI OpenVMS x86-64 V9.2-2:

- VSI TCP/IP Services
- VSI Kerberos
- VSI SSL111
- VSI SSL3
- VSI OpenSSH
- VMSPORTS x86VMS PERL534 T5.34-0

1.14. ZIP/UNZIP Tools

VSI provides the Freeware executables for managing ZIP archives on OpenVMS x86-64 systems. In VSI OpenVMS x86-64 V9.2-2, the installation procedure automatically puts these files in the following directories on the system disk:

Files	Folder
UNZIP.EXE UNZIPSFX.EXE UNZIPSFX_CLI.EXE UNZIP_CLI.EXE UNZIP_MSG.EXE	SYS$COMMON:[SYSHLP.UNSUPPORTED.UNZIP]
ZIP.EXE ZIPCLOAK.EXE ZIPNOTE.EXE ZIPSPLIT.EXE ZIP_CLI.EXE ZIP_MSG.EXE	SYS$COMMON:[SYSHLP.UNSUPPORTED.ZIP]

Chapter 2. New in This Release

This chapter lists the features and bug fixes introduced in the current release, as well as known issues and limitations.

2.1. Fixed Issues

2.1.1. Linking No Longer Fails for C++ Object Files

Previously, the linking process for some of the C++ object files with templates (group sections) would fail with an ACCVIO error in the linker code. This issue has been fixed in the current release.

2.1.2. OpenVMS Librarian No Longer Fails to Insert C++ Object Modules Into a Library

Previously, there was a possibility that the OpenVMS Librarian utility would fail to insert C++ object modules into an object library (.OLB) file, displaying the error:

```
%LIBRAR-F-INSERTERR, error inserting <module-name> in <library-name>
```

This issue has been fixed in the current release.

2.2. Known Issues and Limitations

2.2.1. CLUE$STARTUP Incorrectly Processes Lists of Possible DOSD Devices

Currently, `CLUE$STARTUP` can only process the first Dump Off System Disk (DOSD) device in a list of DOSD devices.

This issue will be fixed in a future release.

2.2.2. SET DUMP/DEVICE Incorrectly Accepts Non-System Shadowed Disks As Dump Target Devices

Currently, the `SET DUMP_OPTIONS/DEVICE` command allows you to set non-system shadowed disks as target devices for writing the system dump information in case of a system crash. However, if you do so, the dump kernel will *not* be able to locate the dump file after a crash, and the system dump information will *not* be written.

Note that this restriction only applies to non-system shadowed disks. The dump kernel can successfully write dumps to a shadowed *system* disk. In a future patch kit, the `SET DUMP_OPTIONS/DEVICE` will be reworked to not accept non-system shadowed disks as dump targets.

2.2.3. Features Not Available in VSI OpenVMS x86-64 V9.2-2

The following functionalities, products, and commands are *not* available in VSI OpenVMS x86-64 V9.2-2:

- ACME_SERVER (only LOGIN82 (SYSUAF) authentication is available)

- Availability Manager

- Process swapping (see Section 3.3.9 of this document)

- RAD support

- Support for privileged applications, such as:

  - User-written device drivers

  - Code that directly calls internal system routines (such as those that manage page tables)

- TECO Editor

2.2.4. DECnet Over Wi-Fi

DECnet Phase IV will not work over a Wi-Fi connection. To be able to use DECnet Phase IV on an x86-64 virtual machine and communicate with other systems, you must have the host machine connected via an Ethernet cable.

DECnet Plus will work over a Wi-Fi connection, but it must be tunnelled through TCP/IP.

2.2.5. Certain Files Might Update Incorrectly

Before upgrading to VSI OpenVMS V9.2-2, check for multiple versions of SYS$HELP:HELPLIB.HLB and SYS$LIBRARY:STARLET.OLB on your disk. If more than one version of either file exists, there is a chance that the latest version will be lost during the upgrade. To avoid this, purge the files so that only one version remains.

This issue will be fixed in a future release of VSI OpenVMS.

2.2.6. Attempts to Add an Existing Proxy as Non-Default May Break Security Server

Any attempt to assign an existing default proxy record as non-default in a network proxy database where the proxy thread is running will render the Security Server unresponsive.

This issue will be fixed in a future release of VSI OpenVMS x86-64.

2.2.7. MSCP-Served Disks May Fail to Mount During System Startup

When mounting MSCP-served disks during system startup, it may take the system some time to bring all the devices online. To determine whether a device has been configured and therefore can be mounted, VSI suggests using the lexical function F$GETDVI(*<disk-name>*, "EXISTS") where *<disk-name>* is the name of the device that you wish to mount. This function should be included in a loop with a short wait between checks and a limit on the number of times through the loop (to skip devices that do not appear in a reasonable amount of time). Once the device has been confirmed to exist, it can be mounted as part of the same script.

2.3. Cross-Tools Kit Update

With VSI OpenVMS x86-64 V9.2-2, use the new **V9.2-2_XGLO** cross-tools kit (VSI-I64VMS-X86_XTOOLS-V0902-2_XGLO-1.ZIP).

For details on the changes to the cross-tools in the **V9.2-2_XGLO** kit, refer to the release notes bundled with the kit. Before the cross-tools kit installation, the release notes file can be extracted from the PCSI kit with the following command:

```
$ PRODUCT EXTRACT RELEASE_NOTES X86_XTOOLS /LOG [/SOURCE=DDCU:[DIR]]
```

Refer to the *VSI OpenVMS x86-64 Cross-Tools Kit Installation and Startup Guide [https://docs.vmssoftware.com/vsi-x86-64-cross-tools-kit-installation-and-startup-guide-v922/]* for complete information on installing the cross-tools kit.

2.4. OpenVMS Clusters on Virtual Machines

VSI OpenVMS x86-64 V9.2-2 supports VirtualBox, KVM, and VMware virtual machines in OpenVMS clusters. Virtual disks can be shared in a cluster via MSCP. Direct (non-MSCP) shared disk access and cluster common system disks are not supported.

VSI OpenVMS x86-64 V9.2-2 has been tested in a clustered configuration using V8.4-1H1, V8.4-2, V8.4-2L1, V8.4-2L2, and V8.4-2L3. VSI has tested 2-node and 3-node clusters with MSCP-served disks where appropriate, CLUSTER_CONFIG_LAN.COM, and many relevant SET, SHOW, and SYSMAN commands. Other configurations will be tested at a later date.

Adding a Node Using a Copy of an Existing System Disk

On VSI OpenVMS x86-64 systems, you must perform an additional step if you use a copy of an existing system disk as the initial system disk of a new node that is being added to a cluster.

In addition to tasks such as modifying the SCSNODE and SCSSYSTEMID parameters and changing the label on the new system disk, you must also change the label for the memory disk. Follow these steps:

Note

The following steps assume that the new system disk is DKA300:, and it is already mounted.

---

1. Invoke LD$STARTUP.COM by using the following command:

   ```
   @SYS$STARTUP:LD$STARTUP.COM
   ```

2. Connect and mount the memory disk container file using the following commands:

   ```
   $ LD CONNECT DKA300:[VMS$COMMON.SYS$LDR]SYS$MD.DSK LDM LDDEV
   $ MOUNT/OVER=ID LDDEV
   ```

3. Note the label of the memory disk. It will be of the form "MD20345927FD". Change the last letter to create a unique name. For example:

   ```
   $ SET VOLUME LDDEV /LABEL=MD20345927FE
   ```

---

4. Dismount the memory disk before completing the other setup tasks for the new system disk.

```
$ DISMOUNT LDDEV
$ LD DISCONNECT LDDEV
```

2.5. Privileged Images Linked Using /SYSEXE Should Be Relinked

VSI recommends that any privileged images linked using the `LINK/SYSEXE` command in VSI OpenVMS V9.2 or earlier be relinked for the current release because data structures and interfaces are subject to change between releases.

Note that the images linked using `LINK/SYSEXE` in VSI OpenVMS V9.2-1 do not need to be relinked for the V9.2-2 release.

2.6. Symmetric Multiprocessing (SMP)

VSI OpenVMS x86-64 V9.2-2 supports a maximum of 32 CPUs.

If you increase the number of CPUs in your virtual machine configuration, you will see messages like the following during system startup:

```
%SMP-I-CPUTRN, CPU #2 has joined the active set.
%SMP-I-CPUTRN, CPU #1 has joined the active set.
%SMP-I-CPUTRN, CPU #3 has joined the active set.
```

Once VSI OpenVMS x86-64 is up and running, the DCL command `SHOW CPU` will reflect your CPU count. For example:

```
$ show cpu
System: X86VMS, VBOX    VBOXFACP
CPU ownership sets:
   Active           0-3
   Configure        0-3
CPU state sets:
   Potential        0-3
   Autostart        0-3
   Powered Down     None
   Not Present      None
   Hard Excluded    None
   Failover         None
$
```

The DCL command `STOP/CPU n` will remove a CPU from the set of CPUs being used. For example:

```
$ stop/cpu 3
%SMP-I-CPUTRN, CPU #3 was removed from the active set.
```

The DCL command `START/CPU n` will add a CPU to the set of CPUs being used. For example:

```
$ start/cpu 3
%SMP-I-CPUTRN, CPU #3 has joined the active set.
```

Chapter 3. In Previous Releases

This chapter lists the features, bug fixes, issues, and limitations introduced in earlier versions of VSI OpenVMS x86-64.

3.1. Fixed Issues

3.1.1. Display of License Charge Information for x86-64 Nodes

Previously, in a cluster with x86-64 nodes running VSI OpenVMS x86-64 V9.2-x and Alpha or Integrity nodes running previous versions of OpenVMS, the SHOW LICENSE/CLUSTER/CHARGE command run from a non-x86-64 node would display the existing x86-64 nodes but would *not* display the license charge information for x86-64 systems.

This issue has been fixed in the following patch kits:

* VMS842L3I_LMF-V0200.ZIP (Integrity)

* VMS842L2A_LMF-V0200.ZIP (Alpha)

3.2. Known Issues and Limitations

3.2.1. BACKUP/INITIALIZE to a Disk Mounted /FOREIGN Does Not Work

A BACKUP command of the form:

```
$ BACKUP/INITIALIZE input-disk:[directories...]*.*; output-disk:save-
set.bck/SAVE
```

where the output volume is a disk that has been mounted /FOREIGN, does *not* work in VSI OpenVMS x86-64 V9.2-2 and may yield the following error:

```
%BACKUP-F-OPENOUT, error opening DKA200:[000000]DKA200$504.BCK; as output
-SYSTEM-W-BADIRECTORY, bad directory file format
```

This type of operation was originally developed to allow the backup of a large non-removable disk onto a series of smaller removable disks. The feature, referred to as "sequential disk" operation, is described in the *VSI OpenVMS System Manager's Manual, Volume 1: Essentials* [https://docs.vmssoftware.com/vsi-openvms-system-manager-s-manual-volume-1-essentials/#SAVESETS_SEQDISK_BCK].

As a workaround, initialize and mount an output volume of a size sufficient to hold the entire backup save set before issuing the BACKUP command as in the following example:

```
$ INITIALIZE output-disk: label
$ MOUNT output-disk: label
$ CREATE/DIRECTORY output_disk:[]
$ BACKUP input-disk:[directories...]*.*; output-disk:save-set.bck/SAVE
```

If a sufficiently large output disk is not available, you can instead create and mount a volume set using multiple disks with the `INITIALIZE` and `MOUNT/BIND` commands.

3.2.2. ENCRYPT Utility Does Not Work as Expected

Most operations with the ENCRYPT utility return the following error:

```
%ENCRYPT-F-ILLALGSEL, algorithm selection unknown, unavailable, or unsupported
```

This issue will be addressed in a future release of VSI OpenVMS x86-64.

3.2.3. Running x86-64 Images on Integrity Systems Causes an Access Violation

When you run a VSI OpenVMS x86-64 image on VSI OpenVMS for Integrity Servers, no message from the image activator appears, but an access violation occurs.

This issue will be corrected in a future patch kit for VSI OpenVMS for Integrity Servers.

3.2.4. Connecting to a Shared LD Container in a Mixed-Architecture Cluster

In OpenVMS x86-64 V9.2-2, a container file can only be connected as a shared LD device accessible to multiple architectures as follows:

1. Connect to the file on all OpenVMS Alpha and/or OpenVMS Integrity systems using a command such as:

   ```
   $ LD CONNECT/alloclass=1/share DISK$LDTEST:[LD]SHARED_LD.DSK LDA5:
   ```

2. Once all required connections on OpenVMS for Alpha and/or OpenVMS for Integrity systems are complete, you may then connect to the file on any OpenVMS x86-64 systems.

   If you connect to the file on an OpenVMS x86-64 system first, any subsequent attempts to connect on an OpenVMS for Alpha and/or OpenVMS for Integrity system will fail with an error message such as:

   ```
   %LD-F-FILEINUSE, File incompatible connected to other LD disk in cluster
   -LD-F-CYLINDERS, Cylinders mismatch
   ```

   VSI intends to provide an update for OpenVMS for Alpha and OpenVMS for Integrity systems. That update will allow the connections to the shared container file to be performed in any order.

3.2.5. Cluster Nodes Running VSI OpenVMS V9.2 May Cause All x86-64 Cluster Members to Crash

In a mixed-version cluster where at least one node is running VSI OpenVMS x86-64 V9.2, certain conditions might trigger a crash of all x86-64 systems, regardless of which OpenVMS version they run. This can happen when a V9.2 MSCP node that is serving disks to other cluster members gets rebooted. In this case, other x86-64 cluster members will see the disks served by MSCP go into mount verification. Upon the verification completion, the MSCP client node might crash.

The fix for this problem is to upgrade all x86-64 cluster nodes to VSI OpenVMS x86-64 V9.2-1 or later.

3.2.6. System Crashes During SYSINIT When Booting With the DEVELOPER Boot Flag

The DEVELOPER boot flag 0x08000000 is reserved for use by VSI, and its function is subject to change. Booting with this flag set can result in an unexpected system behavior.

3.3. Miscellaneous Notes

The notes in this section list the features available on VSI OpenVMS x86-64 compared to the versions of OpenVMS for other architectures.

3.3.1. Additional Prompt During OpenVMS x86-64 Installation

During VSI OpenVMS x86-64 installation, if you choose to install DECnet Phase IV for OpenVMS x86-64 or TCP/IP Services for OpenVMS x86-64, you will see an output similar to the following:

```
* Product VSI X86VMS TCPIP V6.0-23 requires a system reboot.
Can the system be REBOOTED after the installation completes? [YES]
```

Note

The product named in the message may be either DECnet Phase IV or TCP/IP Services.

If this happens, you must answer YES (the default response), otherwise the installation will be terminated.

Later in the installation, you will see the following messages:

```
%PCSI-I-SYSTEM_REBOOT, executing reboot procedure ...

Shutdown/reboot deferred when this product is installed as part of the O/S
installation/upgrade
```

These messages may safely be ignored.

If you are installing both optional products, you will see these messages twice.

VSI will address this issue in a future release.

3.3.2. Extended File Cache (XFC)

VSI OpenVMS x86-64 has extended file caching (XFC) enabled by default.

3.3.3. HYPERSORT Utility Available

The high-performance Sort/Merge utility (HYPERSORT) is available in VSI OpenVMS x86-64. Enable the utility with the following command:

```
$ DEFINE SORTSHR SYS$LIBRARY:HYPERSORT.EXE
```

3.3.4. LIB$INITIALIZE Handling in the Linker

Programs that use the LIB$INITIALIZE startup mechanism must declare a LIB$INITIALIZE PSECT and include the LIB$INITIALIZE module from STARLET.OLB when linking. Traditionally, besides the PSECT, source programs simply declared an external reference to that module, and the linker resolved the reference from STARLET.OLB. However, the LLVM backend used by the cross-compilers removes that external reference from the object file since there were no additional source references to the routine.

The linker was changed to automatically include the required module if it encounters a LIB$INITIALIZE PSECT. For details, see the *VSI OpenVMS Linker Utility Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/].

This change does not affect any source module where external references to the LIB$INITIALIZE module were declared. This change also does not affect any existing link commands that explicitly include the LIB$INITIALIZE module from STARLET.OLB.

3.3.5. Linker: Informational Messages

When the linker encounters writable code sections, with PSECT attributes set to WRT and EXE, it prints the following informational message:

```
%ILINK-I-MULPSC, conflicting attributes for section <PSECT name>
        conflicting attribute(s): EXE,WRT
        module: <module name>
        file: <obj-or-olb-filename>
```

When the linker finds unwind data in a module, but no section with the PSECT attribute set to EXE, it prints the following informational message:

```
%ILINK-I-BADUNWSTRCT, one or more unwind related sections are
missing or corrupted
        section: .eh_frame, there is no non-empty EXE section
        module: <module name>
        file: <obj-or-olb-filename>
```

These messages are seen mainly with MACRO-32 and BLISS source modules. All code sections must be non-writable. You must have code in sections with the PSECT attribute set to EXE.

3.3.6. Handling of Threaded Applications in Linker

When linking applications that use the POSIX Threads Library (PTHREAD$RTL), you can set up the application in such a way that it can receive upcalls from VMS and/or that VMS should map user threads to multiple kernel threads. This behavior can be enabled with the `/THREADS_ENABLE` qualifier. However, if that qualifier is not specified, the linker automatically enables upcalls and displays an informational message to make the user aware of that. The user can overwrite the default behavior by expliclitly specifying `/NOTHREADS_ENABLE`.

3.3.7. Different Image Layout on x86-64 and Itanium

When porting an application from Itanium to x86-64, be aware that the image layout may change in an incompatible way – although the compile and link commands/options did not change. This is an architectural difference.

On Itanium, the compiler may generate **short data** which is accessed in an efficient way. The Itanium linker always collects short data to the DEFAULT CLUSTER, no matter where the object module that defines this short data is collected. That is, in a partially protected shareable image, an object module may be collected into a protected linker cluster, but its short data may be collected into an unprotected cluster, and so it is not protected. User-mode code in the shareable image can write to it.

On x86-64, there is no short data. All data defined in an object module will go where the module goes (except the defining PSECT which is moved with an explicit COLLECT option). That is, on x86-64, for partially protected shareable images, all data defined by an object module which is collected into a protected linker cluster will be protected. User-mode code in the shareable image cannot write to it.

3.3.8. Memory Disks

If you change anything that affects the boot path or dumping, you must run the command procedure SYS$MD.COM before rebooting. For instance, if you change any files referenced or loaded during booting (up to and including the activation of STARTUP), or any files used by the dump kernel, then you must run SYS$MD.COM.

However, in VSI OpenVMS x86-64 V9.2-x there are two exceptions to the above statement. If you make any of the following changes that affect the boot path or dumping, you do not need to run SYS$MD.COM:

1. Use `SYSGEN WRITE CURRENT` or `SYSMAN PARAM WRITE CURRENT`. These commands will access the parameter file on the memory disk directly.

2. Copy a file directly to the memory disk when specifically advised by VSI support specialists to do so.

Use the following command exactly as specified here:

```
$ @SYS$UPDATE:SYS$MD
```

No parameters are needed, the defaults should apply.

When SYS$MD.COM completes, you must reboot.

When SYS$MD.COM is invoked, the system will display something like the following:

```
$ @SYS$UPDATE:SYS$MD
        Created memory disk DKE100:[VMS$COMMON.SYS$LDR]SYS$MD.DSK;1
        - using 184064 blocks in 1 extent with 1010 spare blocks
        - mounted on LDM9323: with volume label MD230120DD6A
        - contains OpenVMS V9.2-1

$
```

After the next system reboot, purge the memory disk with the following command:

```
$ PURGE SYS$LOADABLE_IMAGES:SYS$MD.DSK
```

3.3.9. VSI OpenVMS x86-64 Will Not Support Swap Files

VSI OpenVMS x86-64 does not support swap files. The system's physical memory should be managed with appropriately sized system memory and page file(s).

The AUTOGEN and SWAPFILES procedures no longer create swap files on the system disk. If a swap file resides on the system disk, it will no longer be installed as part of the system startup.

In the current release, the `SYSGEN INSTALL` command does not support the `/SWAPFILE` qualifier. The use of the qualifier will result in a syntax error.

Processes may be seen in the Computable Outswapped (COMO) state. This is a transient state for newly created processes. Processes will never appear in the Local Event Flag Wait Outswapped (LEFO) or Hibernate Outswapped (HIBO) states. All performance counters associated with swapping are still present in the system. Various MONITOR displays will show swapping metrics.

3.3.10. Process Dumps

VSI OpenVMS x86-64 provides support for process dumps. The only method currently available for analyzing process dumps is using the System Dump Analyzer (SDA). Most SDA commands that display data about a process can be used to examine the state of the process. For example, SHOW PROCESS, SHOW CRASH, SHOW EXCEPTION, SHOW CALL, EXAMINE, MAP. Support for the Symbolic Debugger interface will be added in a future release of VSI OpenVMS x86-64.

3.3.11. SYSGEN Parameter Changes

The following changes and additions have been made to the SYSGEN Utility for VSI OpenVMS x86-64. For more information about SYSGEN qualifiers and parameters, see *VSI OpenVMS System Management Utilities Reference Manual, Volume II: M–Z* [https://docs.vmssoftware.com/vsi-openvms-system-management-utilities-reference-manual-volume-ii-m-z/#d0e27590].

Table 3.1. SYSGEN Commands Used for VSI OpenVMS x86-64

Command	Parameter	Description
USE	CURRENT	Specifies that source information is to be retrieved from the current system parameter file on disk. On OpenVMS x86-64 systems, the system parameter file is SYS$SYSTEM:X86_64VMSSYS.PAR.
WRITE	CURRENT	Specifies that source information is to be written to the current system parameter file on disk. The new values will take effect the next time the system is booted. On OpenVMS x86-64 systems, command modifies the current system parameter on disk, SYS$SYSTEM:X86_64VMSSYS.PAR.

Table 3.2. System Parameters

Parameter	Description
BOOT_BITMAP1	On x86-64 systems, this parameter defines the required size in megabytes of the first boot-time allocation bitmap used by SYSBOOT during the bootstrap process on x86-64. If this value is too small, the system may be unable to boot.
BOOT_BITMAP2	On x86-64 systems, this parameter defines the required size in megabytes of the second boot-time allocation bitmap used by SYSBOOT during the bootstrap process on x86-64. If this value is too small, the system may be unable to boot.
DISABLE_X86_FT	On x86-64 systems, DISABLE_X86_FT is a bit mask used to inhibit the use of certain x86 processor features by the operating system.

Parameter	Description
	It is used to decide which variant of the SYSTEM_PRIMITIVES execlet gets loaded. Setting all bits (disabling the use of all optional features) results in SYSTEM_PRIMITIVES_0 being loaded. The following bits are defined: <table><tr><td>**Bit**</td><td>**Definition**</td></tr><tr><td>0</td><td>If 1, do not use the XSAVEOPT instruction.</td></tr><tr><td>1</td><td>If 1, do not use the RDFSBASE, WRFSBASE, RDGSBASE, or WRGSBASE instructions.</td></tr><tr><td>2</td><td>If 1, do not provide software mitigation against the Intel MDS vulnerabilities.</td></tr></table> DISABLE_X86_FT is a STATIC parameter.
GH_EXEC_CODE_S2	On x86-64 systems, GH_EXEC_CODE_S2 specifies the size in pages of the execlet code granularity hint region in S2 space. GH_EXEC_CODE_S2 has the AUTOGEN and FEEDBACK attributes.
GH_EXEC_DATA_S2	On x86-64 systems, GH_EXEC_DATA_S2 specifies the size in pages of the execlet data granularity hint region in S2 space. GH_EXEC_DATA_S2 has the AUTOGEN and FEEDBACK parameters.
GH_RES_DATA_S2	On x86-64 systems, GH_RES_DATA_S2 specifies the size in pages of the resident image data granularity hint region in S2 space. GH_RES_DATA_S2 has the AUTOGEN and FEEDBACK attributes.
GH_RES_CODE	This parameter now applies to x86-64 systems. On x86-64, Integrity, and Alpha systems, GH_RES_CODE specifies the size in pages of the resident image code granularity hint region in S0 space. GH_RES_CODE has the AUTOGEN and FEEDBACK attributes.
GH_RO_EXEC_S0	On x86-64 systems, GH_RO_EXEC_S0 specifies the size in pages of the read-only execlet data granularity hint region in S0 space. GH_RO_EXEC_S0 has the AUTOGEN and FEEDBACK attributes.
GH_RO_RES_S0	On x86-64 systems, GH_RO_RES_S0 specifies the size in pages of the read-only resident image data granularity hint region in S0 space. GH_RO_EXEC_S0 has the AUTOGEN and FEEDBACK attributes.
LOAD_SYS_IMAGES	This special parameter is used by VSI and is subject to change. Do not change this parameter unless VSI recommends that you do so.

Parameter	Description
	LOAD_SYS_IMAGES controls the loading of system images described in the system image data file, VMS$SYSTEM_IMAGES. This parameter is a bit mask. The following bits are defined: <table><tr><td>**Bit**</td><td>**Description**</td></tr><tr><td>0 (SGN$V_LOAD_SYS_IMAGES)</td><td>Enables loading alternate execlets specified in VMS $SYSTEM_IMAGES.DATA.</td></tr><tr><td>1 (SGN$V_EXEC_SLICING)</td><td>Enables executive slicing. Note that executive slicing is always enabled on x86-64 systems.</td></tr><tr><td>2 (SGN$V_RELEASE_PFNS)</td><td>Enables releasing unused portions of granularity hint regions on Alpha servers.</td></tr></table> These bits are on by default. Using conversational bootstrap exec slicing can be disabled. LOAD_SYS_IMAGES is an AUTOGEN parameter.
RAD_SUPPORT	RAD_SUPPORT enables RAD-aware code to be executed on systems that support Resource Affinity Domains (RADs). On x86-64 systems, the default, minimum, and maximum values for RAD_SUPPORT are all zeros because RAD support is not currently available on that platform.
SCSBUFFCNT	SCSBUFFCNT is reserved for VSI use only. On x86-64, Alpha, and Integrity servers, the system communication services (SCS) buffers are allocated as needed, and SCSBUFFCNT is not used.
VCC_FLAGS	The static system parameter VCC_FLAGS enables and disables file system data caching. If caching is enabled, VCC_FLAGS controls which file system data cache is loaded during system startup. <table><tr><td>**Value**</td><td>**Description**</td></tr><tr><td>0</td><td>Disables file system data caching on the local node and throughout the OpenVMS Cluster. In an OpenVMS Cluster, if caching is disabled on any node, none of the other nodes can use the extended file cache or the virtual I/O cache. They cannot cache any file data until that node either leaves the cluster or reboots with VCC_FLAGS set to a non-zero value.</td></tr><tr><td>1</td><td>Enables file system data caching and selects the Virtual I/O Cache.</td></tr></table>

Parameter	Description	
		This value is relevant only for Alpha systems.
	2	Enables file system data caching and selects the extended file cache.
	Note On x86-64 and Integrity servers, the volume caching product [SYS$LDR]SYS$VCC.EXE is not available. XFC caching is the default caching mechanism. Setting the VCC_FLAGS parameter to 1 is equivalent to not loading caching at all or to setting VCC_FLAGS to 0. VCC_FLAGS is an AUTOGEN parameter.	

All system parameters are exposed on every platform: x86-64, Integrity, and Alpha. In addition, flags can be set or cleared on any platform using the SYSGEN Utility. However, the flag may not have any effect on a platform for which it is not intended.

3.3.12. System Crash Dumps

VSI OpenVMS x86-64 system crash dumps are written using a minimal VMS environment called the Dump Kernel. All files used by the Dump Kernel are included in the Memory Disk described in Section 3.3.8.

Interleaved Dumps

Starting with the V9.2-1 release, VSI OpenVMS x86-64 supports two system crash dump types: Compressed Selective Dump and Interleaved Dump.

A Compressed Selective Dump is written using only the primary CPU running in the Dump Kernel, while an Interleaved Dump makes use of the available secondary CPUs. If the dump device is a Solid State Disk (SSD), the dump can be written much faster, thus allowing the system to be rebooted sooner.

The DUMPSTYLE parameter specifies which dump type is being used. The default value for the parameter is 9 (Compressed Selective Dump). However, if the system has more than one CPU and the SYS$SYSTEM:MODPARAMS.DAT file does not include a value for the DUMPSTYLE parameter, then, when AUTOGEN runs, it will set the value of DUMPSTYLE to 128 (Interleaved Dump).

In OpenVMS x86-64, the only other pertinent bits in the DUMPSTYLE parameter are bit 1 (full console output: registers, stack, and system image layout) and bit 5 (only dump system memory, key processes, and key global pages). Either or both of these bits can be set in addition to the two base values (9 and 128). Bit 2 (dump off system disk) is no longer required.

Dump Off System Disk

Crash dumps can be written to the system disk or to an alternate disk which is specifically designated for this purpose.

Dumps to the system disk are written to SYS$SYSDEVICE:[SYSn.SYSEXE]SYSDUMP.DMP which can be created or extended using the SYSGEN utility.

Dumps to an alternate device can be set up as described in the example below:

1. Create a dump file on the desired device using the SYSGEN utility. In this example, we will use the DKA100: disk.

   ```
   $ RUN SYS$SYSTEM:SYSGEN
   SYSGEN> CREATE DKA100:[SYS0.SYSEXE]SYSDUMP.DMP /SIZE=200000
   SYSGEN> EXIT
   ```

2. Enter the following command:

   ```
   $ SET DUMP_OPTIONS/DEVICE=DKA100:
   ```

   You have set DKA100: as the dump device.

You can view the setting by using the SHOW DUMP_OPTIONS command. The change is effective immediately, a reboot *is not* required.

Changes in SET DUMP_OPTIONS

A new qualifier, /MAXIMUM=CPUS=n, has been added to the SET DUMP_OPTIONS command. This qualifier sets the number of CPUs used by the Dump Kernel when writing an Interleaved Dump. It has no effect when a Compressed Selective Dump is used. By default, the Dump Kernel will use all eligible secondary CPUs that are available in the system, up to a maximum of 10 (including the primary CPU). Eligible CPUs are those that were halted successfully when the system crashed and that did *not* trigger the crash with a MACHINECHK or KRNLSTAKNV Bugcheck.

The maximum number of CPUs that can be specified in the command is also 10.

System Dump Analysis

VSI strongly recommends that the version of SDA.EXE and SDA$SHARE.EXE used to analyze a system dump be exactly the same as the version of OpenVMS that was in use when the system crash occurred. However, it is often possible to use SDA images from a different version of OpenVMS, provided there are no major differences between the versions and the warning messages by SDA are ignored (either %SDA-W-LINKTIMEMISM, or %SDA-W-SDALINKMISM, or both).

New SDA command qualifiers in OpenVMS x86-64

The SDA commands below now have new qualifiers that are specific to OpenVMS x86-64. Also, the format of several output messages has been changed to accommodate the architectural differences.

SHOW DUMP command

The qualifier /PERFORMANCE has been added to display the performance data collected by the Dump Kernel while writing a dump.

Depending on the additional keywords used, the data displayed by the SHOW DUMP/PERFORMANCE command can include system information (/PERFORMANCE=SYSTEM), per-CPU information (/PERFORMANCE=CPU), and per-process information (/PERFORMANCE=PROCESS). If no additional keywords are specified with the command, all information mentioned above will be displayed.

SHOW EXECUTIVE command

The qualifier /SDA_EXTENSION has been added to limit the list of displayed executive images to those that are paired with SDA extensions. For example, SWIS$DEBUG.EXE.

SHOW PAGE_TABLE and SHOW PROCESS /PAGE_TABLE commands

The qualifiers /BPTE, /PDE, /PDPTE, /PML4E, and /PML5E have been added to allow you to display a specific level of page table entries. The default behavior is to display the base page table entry (BPTE).

The qualifier /MODE has been added to display the page tables for a specific mode. The default behavior is to display the page tables for all modes. Valid keywords are: KERNEL, EXECUTIVE, SUPERVISOR, and USER.

SHOW POOL command

The output for the qualifier /RING_BUFFER is now affected by the additional qualifiers /ALL and /S2_NPP. The default behavior for SHOW POOL /RING_BUFFER is to display all entries in the Nonpaged Pool ring buffer. If /S2_NPP is specified, then the ring buffer for S2 pool is displayed. If /ALL is specified, then the contents of both ring buffers are displayed, interleaved by their timestamps.

The qualifier /S2_NPP, when used without /RING_BUFFER, displays only the S2 pool. The default behavior is to display all pool types.

The qualifier /USB displays only the nonpaged pool that is reserved for use by USB devices. The default behavior is to display all pool types.

VALIDATE POOL command

The qualifier /S2_NPP allows validation of only the S2 pool. The default behavior is to validate all pool types.

The qualifier /USB allows validation of only the nonpaged pool that is reserved for use by USB devices. The default behavior is to validate all pool types.

SDA commands and qualifiers not available in VSI OpenVMS x86-64

The following commands and qualifiers are not applicable to VSI OpenVMS x86-64 systems:

- SHOW GALAXY

- SHOW GCT

- SHOW GLOCK

- SHOW GMDB

- SHOW SHM_CPP

- SHOW SHM_REG

- SHOW VHPT

- VALIDATE SHM_CPP

- EVALUATE and EXAMINE, qualifiers /FPSR, /IFS, /ISR, /PFS, and /PSR

- SHOW PAGE_TABLE and SHOW PROCESS /PAGE_TABLE, qualifiers /L1, /L2, /L3, and /SPTW

- SHOW POOL, qualifier /BAP

- `VALIDATE POOL`, qualifier `/BAP`

Other SDA command changes

The `COPY` command qualifiers `/COMPRESS`, `/DECOMPRESS`, and `/PARTIAL` cannot be used with an Interleaved Dump.

3.3.13. Traceback Support

The linker includes sufficient traceback information in the image file for a functional symbolic traceback. As a result, by default, the image file may be larger than in previous versions/updates. This additional debug information is not read by the image activator, so it will not slow down image activation. However, to make image files smaller, the linker was changed to include reduced traceback information. This affects the traceback output, as it no longer prints the routine name. Any other traceback output is unaffected. This feature can be enabled with the LINE_NUMBER keyword for the `/TRACE` qualifier. For details, see the *VSI OpenVMS Linker Manual* [https://docs.vmssoftware.com/vsi-openvms-linker-utility-manual/#LINK_COMM_REF_PART].

Traceback now prints the image name, routine name, and line numbers much like traceback on OpenVMS Alpha and OpenVMS Integrity server systems, with the following differences:

1. Traceback is unable to determine the module name, so instead it prints the "basename" of the source file used to create the module.

2. The position of the values in their respective columns may not line up with the header line.

These differences will be addressed in a future release of VSI OpenVMS x86-64.

3.3.14. Viewing Call Stack in Pthread Debugger

Starting with VSI OpenVMS x86-64 V9.2-1, the call stack can be viewed in the Pthread debugger. To show the call stack, use the `-o C` option in the `threads` command. For example, if the debugger runs in the PTHREAD SDA extension, the command to show the call stack for thread id 3 is the following:

```
SDA> pthread threads -o "C" 3
Process name: SECURITY_SERVER   Extended PID: 0000008F  Thread data:
"threads -o "C" 3"
----------------------------------------------------------------------------
thread 3 (blocked, timed-cond) "Process_Proxy_Task", created by pthread
  Stack trace:
    0xffff83000c83b1a5 (pc 0xffff83000c83b1a5, sp 0x00000000024e3228)
    0xffff83000c87b53a (pc 0xffff83000c87b53a, sp 0x00000000024e3230)
    0xffff83000c858493 (pc 0xffff83000c858493, sp 0x00000000024e32e0)
    0xffff83000c852b04 (pc 0xffff83000c852b04, sp 0x00000000024e33f0)
    0xffff83000c8499a3 (pc 0xffff83000c8499a3, sp 0x00000000024e34d0)
    0xffff83000c844e9a (pc 0xffff83000c844e9a, sp 0x00000000024e3800)
    0x0000000080004ad8 (pc 0x0000000080004ad8, sp 0x00000000024e3900)
    0x0000000080007fe6 (pc 0x0000000080007fe6, sp 0x00000000024e3950)
    0xffff83000c8887df (pc 0xffff83000c8887df, sp 0x00000000024e3bd0)
    0xffff83000c83b0ea (pc 0xffff83000c83b0ea, sp 0x00000000024e3f00)
```

However, the output of the `threads -o u` Pthread debugger command that shows an `SDA SHOW CALL` command cannot yet be used in SDA.

```
SDA> pthread threads -o u 3
Process name: SECURITY_SERVER   Extended PID: 0000008F  Thread data:
```

```
"threads -o u 3"
----------------------------------------------------------------------
thread 3 (blocked, timed-cond) "Process_Proxy_Task", created by pthread
  Unwind seed for SDA SHOW CALL 00000000024e2e40
SDA> SHOW CALL 00000000024e2e40
00000000.024E2E40 is no valid handle for a call stack start of
00000000.00000000
```

3.3.15. VSI DECram for OpenVMS

VSI DECram for OpenVMS, also referred to as a RAMdisk, is fully operational in VSI OpenVMS x86-64.

For details of the DECram disk characteristics and configuration, refer to the *DECram for OpenVMS User's Manual* [https://docs.vmssoftware.com/vsi-decram-for-openvms-users-manual/].

3.3.16. Symbolic Links and POSIX Pathname Support

Symbolic links (symlinks) and POSIX pathnames are documented in Chapter 13 of the *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/#SYMLINK_CHAP]. The release notes in this section augment that documentation.

3.3.16.1. Device Names in the POSIX Root

The POSIX file namespace begins at a single root directory. The location of the POSIX root in OpenVMS is defined by the system manager using the `SET ROOT` command. Files may be located via a path starting in the root using an absolute pathname that starts with the `/` character. For example, `/bin` identifies the **bin** directory located in the POSIX root directory. Additionally, all disk devices on an OpenVMS system may be located by using their device name as a name in the POSIX root. For example, the path `/DKA0/USER` identifies the directory DKA0:[USER]. The name after the `/` character may be an actual device name or a logical name that resolves to a device name (and possibly one or more directory names). Device names are not actually present in the POSIX root directory. In resolving an absolute pathname, OpenVMS first searches for the name in the POSIX root. If it is not found, it tries to locate the name as a device or logical name.

3.3.16.2. /SYMLINK Qualifier in DCL Commands

A number of DCL commands that operate on files accept the `/SYMLINK` qualifier to control whether the command operates on a file that a symlink points to or on the symlink itself, and whether symlinks are followed in wildcard searches. For more information, refer to *VSI DCL Dictionary: A–M* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-a-m/] and *VSI DCL Dictionary: N–Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/].

Most commands require `/SYMLINK` to be used with a keyword. The valid keywords are as follows:

Keyword	Explanation
NOWILDCARD	Indicates that symlinks are disabled during directory wildcard searches.
WILDCARD	Indicates that symlinks are enabled during directory wildcard searches.
NOELLIPSIS	Indicates that symlinks are matched for all wildcard fields except for ellipsis.

Keyword	Explanation
ELLIPSIS	Equivalent to WILDCARD (included for command symmetry).
NOTARGET	Indicates that the command operates on the named file whether it is an ordinary file or a symlink.
TARGET	Indicates that if the named file is a symlink, the symlink is followed to operate on the symlink target.

Some commands, such as COPY, DIRECTORY, DUMP, or SET FILE, accept /SYMLINK with no keyword value. In such cases, the qualifier causes the command to operate on the symlink; by default, the command operates on the file pointed to by the symlink.

3.3.16.3. Symlink Support in COPY and CREATE

If the input file of a COPY command is a symlink and the /SYMLINK qualifier is specified, the command copies the symlink; otherwise, it copies the target of the symlink. If the output named in a COPY or CREATE command is an existing symlink, COPY creates a file as identified by the target name of the symlink. Thus, it is possible to create a symlink that points to a non-existent file and then create the file by specifying the symlink as the output of the command.

3.3.16.4. Symlink Support in RENAME

The RENAME command always operates on the file specified in the command. That is, if the input file is a symlink, the symlink is renamed. If a symlink corresponding to the output name exists, it will not be followed.

3.3.16.5. Symlink Support in BACKUP

The BACKUP command never follows symlinks and has no /SYMLINK qualifier. If the input file is a symlink, the symlink is saved or copied. When a save set is restored, BACKUP does not follow symlinks named as output files – instead, the specified name is created. Also, BACKUP does not follow symlinks in its directory wildcarding operation. Any symlinks encountered during directory searches are saved or copied as symlinks.

3.3.16.6. Symlinks and File Versions

A symlink, while implemented as a type of file, may not have multiple versions. Depending on the usage, commands that create files (such as COPY, BACKUP, and RENAME) may attempt to create a new version of a symlink or create a symlink where one or more versions of a file exist. Operations of this kind fail with the following error:

```
NOSYMLINKVERS, cannot create multiple versions of a symlink
```

3.3.16.7. Symlinks Pointing to Multiple File Versions

Even though a symlink is limited to a single file version, it may point to a file that has multiple versions. When a DCL command that searches for multiple files (such as DIRECTORY or SET FILE) locates a file via a symlink, it returns the name of the symlink. As a result, even though multiple versions of the symlink target may exist, the DCL command operates only on the latest version of the target file.

For example, in the following sequence of commands:

```
$ CREATE/SYMLINK=FILE.TXT LINK.TXT
```

```
$ COPY FILE2.TXT LINK.TXT
$ COPY FILE2.TXT LINK.TXT
$ COPY FILE2.TXT LINK.TXT
```

three versions of the file FILE.TXT will exist, but a DIRECTORY command will show only the latest version in response to the name LINK.TXT. Likewise, the command $ TYPE LINK.TXT;* will display only the latest version of FILE.TXT, not all three versions.

3.3.17. Symbolic Debugger

VSI OpenVMS x86-64 V9.2-2 includes an initial implementation of the Symbolic Debugger. While many features work, there are some that do not work. These are listed in the following sections. VSI will address these issues in a future release. In addition, some of the missing features will require future native compilers, as the cross compilers are unable to provide sufficient debug information to the debugger.

3.3.17.1. Supported Registers

All integer registers (see the table below) are currently supported, including the 32, 16, and 8-bit variants.

Table 3.3. Supported registers

64-bit registers	%rax, %rbx, %rcx, %rdx, %rsi, %rdi, %rsp, %rbp, %r8, %r9, %r10, %r11, %r12, %r13, %r14, %r15
32-bit registers	%eax, %ebx, %ecx, %edx, %esi, %edi, %esp, %ebp, %r8d, %r9d, %r10d, %r11d, %r12d, %r13d, %r14d, %r15d
16-bit registers	%ax, %bx, %cx, %dx, %si, %di, %sp, %bp, %r8w, %r9w, %r10w, %r11w, %r12w, %r13w, %r14w, %r15w
8-bit registers	%al, %ah, %bl, %bh, %cl, %ch, %dl, %dh, %sil, %dil, %spl, %bpl, %r8b, %r9b, %r10b, %r11b, %r12b, %r13b, %r14b, %r15b

3.3.17.2. Older Versions of Compilers Always Set Language to C

The issue with the x86-64 compilers setting the debug language to C by default has been resolved in the current release of VSI OpenVMS x86-64.

However, older versions of cross-compilers still set the debug language to C by default. This means that when the debugger regains control, the language is set to C, even if the module being debugged was written in another language.

The way to work around this problem is simply to use the SET LANGUAGE command to set the default language to that which is being debugged.

3.3.17.3. Language Support Limitations

There is some support for language-specific features when using the EXAMINE and DEPOSIT commands. However, some of the more complex data structures may not work correctly and can have unexpected and undefined behaviour.

As mentioned previously, the cross-compilers all set the language type to C in the debug output. This may appear to prevent language-specific features from working. Using the SET LANGUAGE command will resolve this.

3.3.17.4. Source Line Correlation

In this release, the debugger fully supports source line correlation. Note, however, that the current version of the debugger requires you to use the latest versions of cross-compilers and/or native compilers.

Previous versions of the debugger *do not* support source line correlation because the previous versions of compilers generate listing line numbers in their debug data, not source lines. In most instances, this will lead to quite a large disparity between the line numbers available to the debugger and the actual line numbers of the source file. This can manifest itself in messages similar to the following:

```
break at DANCE\pause_actions\%LINE 138517
%DEBUG-I-NOTORIGSRC, original version of source file not found
        file used is SYS$SYSDEVICE:[DEBUG.BLUEY]DANCE.C;1
%DEBUG-W-UNAREASRC, unable to read source file SYS$SYSDEVICE:
[DEBUG.BLUEY]DANCE.C;1
-RMS-E-EOF, end of file detected
```

To work around this issue, VSI recommends you use the debugger with accompanying source listings to locate relevant lines reported by break points and the EXAMINE command.

The lack of source line correlation support also means that the STEP/LINE command does not always work correctly and can behave like the STEP/INSTRUCTION command.

3.3.17.5. Floating-Point Support

There is currently no support for floating-point registers. Although it is possible to examine and deposit them, the contents are inaccurate and will not be updated.

3.3.18. User-Written x86-Assembler Modules

User-written x86-assembler code must follow the VSI OpenVMS calling standard (see the *VSI OpenVMS Calling Standard manual* [https://docs.vmssoftware.com/vsi-openvms-calling-standard/#X86_64_CONVENTIONS_CH]) to provide exception handling information that is used by the exception handling mechanism to find a handler in one of the callers of the assembler module. Without that information, exception handling can only call the last chance handler, which means the application will likely not be able to handle the exception.

See the following example of a user-written x86-assembler module:

```
        .vms_module "MYTEST","X-01"
        .text
        .globl  MYTESTRTN
        .align 16
        .type   MYTESTRTN,@function

MYTESTRTN:
        .cfi_startproc
        pushq   %rbp                    // Establish a frame pointer
        .cfi_def_cfa_offset 16          // Record distance to saved PC
        .cfi_offset %rbp, -16
        movq    %rsp,%rbp
        .cfi_def_cfa_register %rbp
```

```
// function body

movq    %rbp,%rsp                // Restore stack and return
popq    %rbp
.cfi_def_cfa %rsp, 8             // Adjust distance to saved PC
retq
.size MYTESTRTN, .-MYTESTRTN
.cfi_endproc
```

3.3.19. CD Audio Functionality Not Supported on x86-64

CD audio functionality is not supported on VSI OpenVMS running on x86-64. CD audio functionality has been deprecated for all x86 platforms and beyond.

This does *not* apply to VSI OpenVMS running on Alpha or Integrity platforms.

3.3.20. STABACKIT.COM Deprecated

SYS$UPDATE:STABACKIT.COM has been been deprecated in VSI OpenVMS x86-64. The STABACKIT functionality originally supported Standalone Backup for VAX systems and has long been replaced by SYS$SYSTEM:AXPVMS$PCSI_INSTALL_MIN.COM for OpenVMS Alpha and SYS$SYSTEM:I64VMS$PCSI_INSTALL_MIN.COM for OpenVMS Integrity systems. The referenced command procedures produce a minimal installation of OpenVMS on a target device which may be booted to allow a full backup of the system disk. STABACKIT would offer to run the appropriate procedure for you.

VSI OpenVMS x86-64 V9.2-2 currently does not support minimal installation of the operating system on a non-system disk. Backup options for virtual machines include the ability to perform backup from the host operating system and the ability to checkpoint or clone within the hypervisor. One may also boot from the installation media and choose the DCL subprocess menu entry to perform a backup of the system device.

3.3.21. SMP Timeout Parameters Increased

Starting with VSI OpenVMS V9.2-1, the default values for the system parameters SMP_SPINWAIT and SMP_LNGSPINWAIT have been increased. This change was made to avoid potential CPUSPINWAIT and CLUEXIT crashes occurring on virtual machines.

3.3.22. Improvements to System Memory Allocation

When allocating large amounts of system memory such as when allocating a large DECram disk, the operation could take a long time. These operations could result in a CLUEXIT or CPUSPINWAIT crash. Starting with VSI OpenVMS V9.2-1, improvements have been made to the system to improve the efficiency of the system memory allocation.

3.3.23. Contiguous Best Try Qualifier for SET FILE/ATTRIBUTES

The `SET FILE/ATTRIBUTES` command supports the Contiguous Best Try (CBT) keyword.

With CBT enabled, the file system will allocate additional blocks to a file contiguously on a best effort basis. The file system will disable the attribute if the best effort algorithm cannot complete the file extension with at most three additional extents.

To enable CBT, use the following command:

```
$ SET FILE/ATTRIBUTES=CBT
```

To disable CBT, use the following command:

```
$ SET FILE/ATTRIBUTES=NOCBT
```

3.3.24. /EXTENTS Qualifier for ANALYZE/DISK_STRUCTURE

The `ANALYZE/DISK_STRUCTURE` command now supports the `/EXTENTS` qualifier.

`ANALYZE/DISK_STRUCTURE/EXTENTS` will produce a report on the fragmentation of free space on a volume. By default, the only output is the number of extents of free space and the total number of free blocks. For additional details, specify one of the following additional qualifiers with the `ANALYZE/DISK_STRUCTURE/EXTENTS` command:

Qualifier	Syntax	Description
/LARGEST	/LARGEST[=*n*]	Displays a list of block counts for the *n* largest extents of free space on the volume in descending size order. The default for *n* is 10. The qualifier is ignored if you specify zero or a negative number. The list is also saved in the DCL symbol ANALYZE$LARGEST_EXTENTS (as a comma-separated list of decimal values). The symbol is set to an empty string if there is no free space on the disk. There is no upper limit on *n*, but if the DCL symbol exceeds 1024 characters, the number of extents in the symbol will be reduced to ensure the symbol is no more than 1024 characters.
/LOCK_VOLUME	/LOCK_VOLUME /NOLOCK_VOLUME	Locks the volume against allocation while the data is being collected. By default, the volume is locked.
/OUTPUT	/OUTPUT[=*filespec*] /NOOUTPUT	Specifies the output file for the fragmentation report produced by the ANALYZE/DISK_STRUCTURE utility. If you omit the qualifier or the entire filename, SYS$OUTPUT will be used. The default filename is ANALYZE$EXTENTS.LIS.
/REQUIRED	/REQUIRED=n	Displays the number of extents required to satisfy an allocation request of *n* blocks (starting with the largest extent). There is no default for *n*. If you specify zero or a negative number, the qualifier is ignored. The result is also saved in the DCL symbol ANALYZE$REQUIRED_EXTENTS. The symbol

Qualifier	Syntax	Description
		is set to an empty string if there is insufficient space on the disk to satisfy the allocation request.

Consider the following example:

```
$ ANALYZE/DISK_STRUCTURE/EXTENTS/LARGEST/REQUIRED=20000 LDM9063:

Extent report for _X86VMS$LDM9063:
=================================

The disk has 6 extents of free space for a total of 25262 free blocks.

The extent sizes are:

                17176
                 5591
                 2469
                   15
                    9
                    2

2 extents are required for an allocation of 20000 blocks.
```

3.3.25. /OPTIONS qualifier for PRODUCT SHOW PRODUCT

The `PRODUCT SHOW PRODUCT` command supports the `/OPTIONS=keyword` qualifier. A keyword is required. Currently, the only defined value is `EXTENDED_DISPLAY`. If you specify `/OPTIONS=EXTENDED_DISPLAY`, the system will output an additional line of information for each of the listed products, giving you the DESTINATION that was specified during the product installation. If no alternative destination was used during the product installation, the system disk will be shown as the destination. See the following example:

```
$ PRODUCT SHOW PRODUCT/OPTIONS=EXTENDED_DISPLAY *VMS
------------------------------------ ----------- ---------
PRODUCT                              KIT TYPE    STATE
------------------------------------ ----------- ---------
VSI X86VMS OPENVMS V9.2-2            Platform    Installed
    Destination: DISK$SYSTEM_DISK:[VMS$COMMON.]
VSI X86VMS VMS V9.2-2               Oper System Installed
    Destination: DISK$SYSTEM_DISK:[VMS$COMMON.]
------------------------------------ ----------- ---------
2 items found
```

The `/OPTIONS` qualifier has been available since V8.4-1H1. It can be combined with other qualifiers, for example, `/FULL`.

3.3.26. CHECKSUM Utility Supports SHA1 and SHA256 Algorithms

In VSI OpenVMS x86-64, the CHECKSUM utility supports the SHA1 and SHA256 secure hash algorithms to calculate file checksums. These algorithms calculate a checksum for all bytes within a file and ignore possible record structures.

Use the CHECKSUM command qualifier /ALGORITHM=option to specify the algorithm for the file checksum calculation.

For information about all supported checksum algorithms, refer to the *VSI OpenVMS DCL Dictionary: A-M* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-a-m/#CHECKSUM].

3.3.27. VSI C Run-Time Library (C RTL) Update

VSI OpenVMS x86-64 V9.2-2 includes an updated VSI C Run-Time Library (C RTL). The update provides bug fixes, as well as new functions, including the additional C99 Standard functions, new constants, new and updated header files.

See Appendix A of this document for more detailed information.

3.3.28. SCS Jumbo Packets Not Enabled on Boot

VSI OpenVMS x86-64 V9.2-x does not enable Jumbo Packets automatically during boot. You can enable Jumbo Packets manually. Note that after you do so, you must restart the LAN adapter communications.

3.3.29. Large Hardware Page Usage

VSI OpenVMS x86-64 V9.2-x takes advantage of 2 MB hardware pages in limited areas of the operating system. Usage of the larger page size allows for faster memory access. Larger pages are used by parts of the executive code and data along with some internal memory management data.

3.3.30. Entropy

VSI OpenVMS x86-64 V9.2-x collects information from stochastic system events and hardware-provided entropy sources, when available, to create a pool of random data which may be used as seeds for random number and cryptographic algorithms. The features associated with entropy collection include the RANDOM_SOURCES SYSGEN parameter and the SYS$GET_ENTROPY system service.

Note

On hardware that supports the Intel RDRAND instruction, entropy collection will include random data from the RDRAND instruction as part of the entropy pool mix. However, in spite of the host hardware supporting RDRAND, not all hypervisors support this instruction, and hypervisor support may be configurable on a per-VM basis. To display the information about the status of the RDRAND instruction, use the new DCL command SHOW ENTROPY (see SHOW ENTROPY).

3.3.30.1. SYSGEN Parameter RANDOM_SOURCES

The RANDOM_SOURCES SYSGEN parameter allows you to select the software and hardware sources of data that will contribute to the entropy pool. The default value is -1, meaning all possible sources are enabled.

Note

Not all categories of sources are implemented at this time. It is expected that additional sources will be added in subsequent updates or operating system releases.

For a detailed desciption of RANDOM_SOURCES, execute the command `HELP SYS_PARAMETERS RANDOM_SOURCES`.

VSI recommends leaving RANDOM_SOURCES set to `-1` unless you are advised differently by VSI support or engineering. RANDOM_SOURCES is a dynamic parameter and *does not* require a reboot after a value change.

3.3.30.2. SYS$GET_ENTROPY System Service

SYS$GET_ENTROPY

SYS$GET_ENTROPY — Returns up to 256 bytes of entropy from the system-wide entropy pool. This service accepts 64-bit addresses.

Format

SYS$GET_ENTROPY buffer, buflen

C Prototype:

`int sys$get_entropy(void * buffer, unsigned __int64 buflen);`

Parameters

BUFFER

A 32- or 64-bit address of a buffer to receive the random data. The service will fill the buffer with random data up to buffer length number of bytes. The maximum amount of data returned in a single call is 256 bytes.

OpenVMS Usage:	address
Type:	address
Access:	write only
Mechanism:	by value

BUFLEN

Size of the buffer in bytes. The maximum value is 256.

OpenVMS Usage:	byte count
Type:	integer
Access:	read only
Mechanism:	by value

Description

The SYS$GET_ENTROPY service retrieves bytes of data from the system entropy pool and writes them to the address specified in the buffer parameter.

Required Privileges

None.

Required Quota

None.

Condition Values Returned

SS$_NORMAL	Successful completion.
SS$_ACCVIO	The supplied buffer is not writeable in the callers mode.
SS$_BADBUFLEN	Buffer length is zero or larger than 256.

3.3.30.3. SHOW ENTROPY

The `SHOW ENTROPY` command provides information about the state of the entropy engine. For more information, see *VSI OpenVMS DCL Dictionary: N-Z* [https://docs.vmssoftware.com/vsi-openvms-dcl-dictionary-n-z/#BRASS_ENTROPY] and the VSI OpenVMS HELP entry for this command.

3.4. Virtualization Notes

The notes in this section describe known issues and limitations when running VSI OpenVMS x86-64 V9.2-2 as a **guest operating system** in Oracle VM VirtualBox, KVM, and VMware virtual machines.

3.4.1. Changing Settings of a Running Virtual Machine May Cause a Hang

Even if the hypervisor appears to allow it, *do not* change the configuration of a VSI OpenVMS virtual machine while it is running, as this may cause it to hang. Before editing the settings, make sure to power off the virtual machine.

3.4.2. Time of Day May Not Be Maintained Correctly in Virtual Machine Environments

VSI OpenVMS x86-64 may not maintain the time of day correctly in virtual machine environments after certain events, such as booting, suspending, taking a snapshot of a virtual machine, or other similar events (depending on the virtual machine host). To keep the time of day accurate in such cases, use the `SET TIME` command. If this does not resolve the problem, enter the following command via the SYSTEM account:

```
$ @SYS$MANAGER:UTC$SET_TIME.COM
```

This issue will be addressed in a future release of VSI OpenVMS x86-64.

3.4.3. System Time on KVM Virtual Machines

On KVM/QEMU systems, when a virtual machine is powered on, the system time is set based on the value of the CLOCK OFFSET parameter in the configuration file of that virtual machine. The default value is `'utc'`. Depending on the physical location of the host system, this might lead to differences between system times of the VM and the host.

To resolve this problem, use the `virsh edit` command to edit the XML configuration file of your virtual machine and change the value of the CLOCK OFFSET parameter to `'localtime'`, like so:

```
<clock offset='localtime'>
```

For more information, see the official documentation for your Linux distribution.

3.4.4. VirtualBox and Hyper-V Compatibility on Windows 10 and 11 Hosts

Host systems running Windows 10 and 11 that have previously run Microsoft Hyper-V hypervisor may fail the CPU feature checks. The issue is that certain CPU features are supported on the host system (the **vmscheck.py** script passes), but not on the guest system (the OpenVMS Boot Manager check fails). Primarily, the XSAVE instruction may not be present on the guest system.

This issue persists even if the Hyper-V feature has been removed. This happens because certain Hyper-V services interfere with VirtualBox.

The VirtualBox developers are aware of this issue and are working to improve the interoperability with Hyper-V.

To explicitly disable execution of the Hyper-V services that interfere with VirtualBox, perform the following steps on your Windows host system:

1. Run Command Prompt as administrator.

2. In Command Prompt, execute the following command to disable Hyper-V:

   ```
   bcdedit /set hypervisorlaunchtype off
   ```

3. Shut down your Windows host system by executing the following command:

   ```
   shutdown -s -t 2
   ```

4. Power on and boot your Windows host system again.

The XSAVE instruction should now be available to your VirtualBox guest.

For more information about the CPU feature checks, see Section 1.4 in the Chapter 1 section.

Tips on How To Determine If Hyper-V Services Impact Your VirtualBox VM

When you launch a VirtualBox guest, look for the icon in the guest window status bar.

* A green turtle icon () indicates that the VirtualBox host is running as a Hyper-V guest with diminished performance.

* An icon with a V symbol () indicates that you are running directly on a VirtualBox host.

View the log file VBOX.LOG.

1. To open the log file, in the VirtualBox VM Manager window, right-click on the virtual machine entry and select **Show Log** from the menu.

2. In the log file, search for "XSAVE".

   * If it shows "1 (1)", your VM guest has XSAVE.

   * If it shows "0 (1)", your VM guest has Hyper-V services impacting it.

3. In the log file, search for "HM". The following message also indicates that Hyper-V is active:

```
{timestamp} HM: HMR3Init: Attempting fall back to NEM: VT-x is not available
{timestamp} NEM: WHvCapabilityCodeHypervisorPresent is TRUE, so this might work.
```

3.4.5. VirtualBox: TCP Ports May Become Unusable After Guest Is Terminated

When running VSI OpenVMS x86-64 as a guest in a VirtualBox VM, TCP console ports may become unusable after a guest session has been terminated. After that, you cannot connect to your VirtualBox VM again. These ports remain in the LISTEN state even after you have disconnected the remote terminal session.

As a workaround, use the following commands to free your TCP ports and connect to your VirtualBox VM:

```
vboxmanage controlvm <vmname> changeuartmode1 disconnected
vboxmanage controlvm <vmname> changeuartmode1 tcpserver <port>
```

The VirtualBox developers have indicated that the fix will be provided in an upcoming VirtualBox maintenance release.

3.4.6. VMware Guest May Fail to Boot After Adding Second SATA Controller
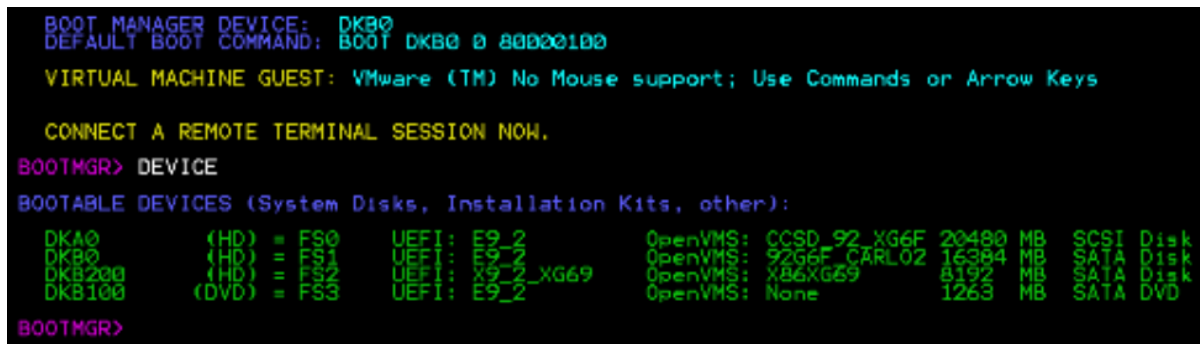
It has been reported that a VMware guest does not boot when a second SATA controller is added to the configuration. In their case, removing the second SATA controller eliminates the issue.

VSI has not observed boot issues when adding a second SATA controller during testing. If you encounter this situation, please report your issue via the VMS Software Service Platform.

3.4.7. Boot Manager Displays Incomplete List of Bootable Devices

If you are running a VMware virtual machine with multiple buses, you may run into a problem with the list of bootable devices displayed by the Boot Manager.

If, after entering the DEVICE command, your Boot Manager display normally looks similar to the following:

but occasionally, upon shutting down your VMware virtual machine, it appears as shown below:

This means, not all of your devices and/or buses have been configured properly. Proceeding to boot your VMware virtual machine from this truncated configuration display may result in an incomplete configuration of your Virtual Machine's buses and the disk devices on those buses.

This happens because, by default, VMware products do not allow the UEFI Shell to be launched by the platform firmware Boot Option and have the Quick Boot option enabled.

These problems can be resolved by setting the correct values for the `efi.shell.activeByDefault` and `efi.quickBoot.enabled` parameters. To do so, follow the procedure described in Section 3.4.8.

3.4.8. Possible Issues with VMware Virtual Machines

Virtual machines created in VMware hypervisors (ESXi, Workstation Pro, Player, Fusion) may not operate as intended until you manually set the parameters listed in the table below.

Depending on your specific configuration, there may be cases where you may not need to set one or more of these parameters. VSI provides this information in case you experience the issues these parameters address.

Key/Parameter Name	Value	Description
efi.serialconsole.enabled	TRUE	This parameter enables serial console access to the UEFI Shell and VSI OpenVMS Boot Manager. By default, VMware disables serial port console access. Because of this, a remote serial connection only becomes active once OpenVMS is booted (SYSBOOT or beyond).
efi.shell.activeByDefault	TRUE	By default, VMware products do not allow the UEFI Shell to be launched by the platform firmware Boot Option. Setting this parameter to TRUE will allow for automatic launching of the Shell.
efi.quickBoot.enabled	FALSE	By default, the **Quick Boot** option is enabled. With **Quick Boot** enabled, the VM attempts to map *only the essential* devices to speed up the boot process. However, to make sure that *all* devices are visible to the VSI OpenVMS Boot Manager, **Quick Boot** must be disabled. Be aware that on large configurations with hundreds of disks, the boot process can take several minutes.

Warning

Note that key names are *case-sensitive*.

To set these parameters for a VMware ESXi virtual machine, follow these steps:

1. Select your VM and click **Edit**.

2. Switch to the VM Options tab.

3. Expand the **Advanced** menu.

4. Under Configuraton Parameters, click **Edit Configuration**.

5. Click **Add Parameter**.

6. Enter the new key (parameter name) and set the value according to the table above.

    > **Note**
    >
    > Quotes around the values are *not* required on ESXi.

7. Click **OK**.

8. Repeat steps 5 through 7 to add the other two parameters.

9. Click **Save**.

To set these parameters for a virtual machine running under any other VMware product, follow these steps:

1. Determine the location of your VM configuration file. To do so, perform the following steps:

    a. Select your VM and bring up its Settings window.

    b. Switch to the **Options** tab.

    c. The directory specified in the **Working Directory** field is where you will find your VM configuration file (it will be named *vm_name.vmx*).

2. Make sure your VM is powered off.

3. Open the folder that contains your VM configuration file.

4. Open the file in an editor.

5. Enter the new keys (parameter names) and set their values according to the table above.

    > **Note**
    >
    > Quotes around the values *are required* when manually editing VMX files.

6. Save and close the file.

3.4.9. One VirtIO-SCSI Adapter Supported on KVM

On KVM/QEMU, only *one* VirtIO-SCSI adapter can be configured and used on a VSI OpenVMS x86-64 V9.2-x system. Note that this functionality requires QEMU version 6.2 or later.

3.5. Layered and Open-Source Products Notes

Layered products and field-test versions of open-source products for VSI OpenVMS x86-64 can be downloaded individually from the VMS Software Service Platform [https://sp.vmssoftware.com/].

Released versions of open-source products for VSI OpenVMS x86-64 can be downloaded from the official VMS Software website [https://vmssoftware.com/products/list/?license=Open%20Source].

VSI recommends that you regularly check both resources for up-to-date versions of layered and open-source products for VSI OpenVMS x86-64.

For detailed information about the products, please refer to the associated product release notes bundled with the kits.

Appendix A. VSI C Run-Time Library (C RTL) Notes

A.1. C99 Update

VSI OpenVMS x86-64 includes the updated C RTL that provides additional C99 Standard functions and functionality that were not previously available.

These functions are also available on the following VSI OpenVMS versions:

- VSI OpenVMS Integrity V8.4-2L1 and V8.4-2L3

- VSI OpenVMS Alpha V8.4-2L1 and V8.4-2L2

To utilize C99 Standard functions, compile your applications with the `/STANDARD=C99`, `/STANDARD=LATEST`, or `/STANDARD=RELAXED` (default) switches. See the section Section A.1.1 for a list of functions.

The value of the __CRTL_VER macro, predefined by the VSI C Compiler, has been changed from 80400000 to 80500000.

Note

If you develop an application on a system with the CRTL 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)`.

This release also includes changes to some header files to make them more consistent with the standards.

MATH.H, FP.H, and FLOAT.H	Definitions have been moved around/between these header files to match the C99 Standard requirements.
STDINT.H and INTTYPES.H	Definitions from INTTYPES.H have been moved into a new header file, STDINT.H, to match the standard's requirements. INTTYPES.H now contains '#include <STDINT.h>' so that existing applications will continue to compile without any changes. In addition, some new names have been defined for data types to match the C99 Standard. For example, int64_t.

Possible errors when compiling applications

With the addition of new data type and function definitions, 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. Conflicting function definitions can result in various %CC errors or warnings. To diagnose such problems, compile the application using the `/LIST/SHOW=INCLUDE` command and then examine the listing file.

There are different ways to resolve such problems. Some examples are following:

- 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
```

Possible informational and warning messages when linking applications

The implementations of isnan() and isnormal() have changed and now utilize functions in the Math Run-Time Library (DPML$SHR.EXE). If your application includes references to isnan() or isnormal() and you encounter the %ILINK-I-UDFSYM and %ILINK-W-USEUNDEF messages for MATH$ symbols when linking your application, you may add SYS$LIBRARY:DPML$SHR/SHAREABLE to your options file as one way of resolving undefined symbolic references.

UNSUPCONVSPEC warning

When using the new format specifiers with print and scan (see Section A.1.1.12), the system will generate a %CC-W-UNSUPCONVSPEC warning.

You can eliminate the warnings by adding #pragma message disable UNSUPCONVSPEC to your code or by compiling your code with the switch, /WARNING=DISABLE=UNSUPCONVSPEC. This warning will be removed in a future update to the C compiler.

Online Help

A future version of VSI OpenVMS x86-64 will update the Online Help contents of the C RTL with the functions listed in this document.

A.1.1. C99 Functions

This section describes the C99 functions that have been added to the C RTL. For VSI OpenVMS x86-64, these functions are included in the C RTL.

For VSI OpenVMS for Integrity and VSI OpenVMS for Alpha systems, these functions are included in the following kits:

- C99 V1.0

- C99 V2.0

- RTL V2.0

A.1.1.1. fpclassify

Format

```
#include <math.h>
int fpclassify (real-floating x);
```

Description

The `fpclassify` macro classifies its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then, the classification is based on the type of the argument.

Returns

The `fpclassify` macro returns the value of the number classification macro appropriate to the value of its argument.

A.1.1.2. isblank, iswblank

Format

```
#include <ctype.h>
int isblank (int c);
#include <wctype.h>
int iswblank (wint_t wc);
```

Description

The `isblank` function tests for any character that is a standard blank character or is one of a locale-specific set of characters for which `isspace` is true and that is used to separate words within a line of text. The standard blank characters are the following: space (' ') and horizontal tab ('\t'). In the "C" locale, `isblank` returns true only for the standard blank characters.

The `iswblank` function tests for any wide character that is a standard blank wide character or is one of a locale-specific set of wide characters for which `iswspace` is true and that is used to separate words within a line of text. The standard blank wide characters are the following: space (L' ') and horizontal tab (L'\t'). In the "C" locale, `iswblank` returns true only for the standard blank characters.

Returns

These functions return true if and only if the value of the character or wide character has the property described in the description.

A.1.1.3. isgreater, isgreaterequal, isless, islessequal, islessgreater, isunordered

Format

```
#include <math.h>
int isgreater (x, y);
int isgreaterequal (x, y);
int isless (x, y);
int islessequal (x, y);
int islessgreater (x, y);
int isunordered (x, y);
```

Description

The normal relation operations (like <, "less than") will fail if one of the operands is NaN. This will cause an exception. To avoid this, C99 defines the macros listed below.

These macros are guaranteed to evaluate their arguments only once. The arguments must be of real floating-point type (note: do not pass integer values as arguments to these macros, since the arguments will not be promoted to real-floating types).

isgreater ()	determines (x) > (y) without an exception if x or y is NaN.
isgreaterequal ()	determines (x) >= (y) without an exception if x or y is NaN.
isless ()	determines (x) < (y) without an exception if x or y is NaN.
islessequal ()	determines (x) <= (y) without an exception if x or y is NaN.
islessgreater ()	determines (x) < (y) ‖ (x) > (y) without an exception if x or y is NaN. This macro is not equivalent to x != y because that expression is true if x or y is NaN.
isunordered ()	determines whether its arguments are unordered, that is, whether at least one of the arguments is a NaN.

Returns

The macros other than `isunordered()` return the result of the relational comparison; these macros return 0 if either argument is a NaN.

`isunordered()` returns 1 if x or y is NaN and 0 otherwise.

A.1.1.4. llrint, llrintf, llrintl

Format

```
#include <math.h>
long long int llrint (double x);
long long int llrintf (float x);
long long int llrintl (long double x);
```

Description

The `llrint` functions round their argument to the nearest integer value, rounding according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and a domain error or range error may occur.

Returns

The `llrint` functions return the rounded integer value.

A.1.1.5. llround, llroundf, llroundl

Format

```
#include <math.h>
long long int llround (double x);
long long int llroundf (float x);
long long int llroundl (long double x);
```

Description

The `llround` functions round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and a domain error or range error may occur.

Returns

The `llround` functions return the rounded integer value.

A.1.1.6. nearbyint, nearbyintf, nearbyintl

Format

```
#include <math.h>
double nearbyint (double x);
float nearbyintf (float x);
long double nearbyintl (long double x);
```

Description

The `nearbyint` functions round their argument to an integer value in floating-point format, using the current rounding direction and without raising the "inexact" floating-point exception.

Returns

The `nearbyint` functions return the rounded integer value.

A.1.1.7. round, roundf, roundl

Format

```
#include <math.h>
double round (double x);
float roundf (float x);
long double roundl (long double x);
```

Description

The round functions round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

Returns

The round functions return the rounded integer value.

A.1.1.8. scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl

Format

```
#include <math.h>
double scalbln (double x, long int n);
float scalblnf (float x, long int n);
long double scalblnl (long double x, long int n);
double scalbn(double x, int n);
float scalbnf(float x, int n);
long double scalbnl(long double x, int n);
```

Description

These functions multiply their first argument *x* by FLT_RADIX (probably 2) to the power of *n*, which is:

```
x * FLT_RADIX ** n
```

The definition of FLT_RADIX can be obtained by including <float.h>.

Returns

On success, these functions return x × FLT_RADIX ** n.

If *x* is a NaN, a NaN is returned.

If *x* is positive or negative infinity, positive or negative infinity is returned.

If *x* is +/- 0, +/- 0 is returned.

If the result overflows, a range error occurs, and the functions return HUGE_VAL, HUGE_VALF, or HUGE_VALL, respectively, with a sign the same as *x*.

If the result underflows, a range error occurs, and the functions return zero, with a sign the same as *x*.

A.1.1.9. strtof, strtold, wcstof, wcstold

Format

```
#include <stdlib.h>
float strtof (const char * restrict nptr, char ** restrict endptr);
long double strtold (const char * restrict nptr, char ** restrict endptr);
#include <wchar.h>
float wcstof (const wchar_t * restrict nptr,
wchar_t ** restrict endptr);
long double wcstold (const wchar_t * restrict nptr,
wchar_t ** restrict endptr);
```

Function Variants

The `strtof` function has variants named `_strtof32` and `_strtof64` for use with 32-bit and 64-bit pointer sizes, respectively. The `strtold` function has variants named `_strtold32` and `_strtold64` for use with 32-bit and 64-bit pointer sizes, respectively. The `wcstof` function has variants named `_wcstof32` and `_wcstof64` for use with 32-bit and 64-bit pointer sizes, respectively. The `wcstold` function has variants named `_wcstold32` and `_wcstold64` for use with 32-bit and 64-bit pointer sizes, respectively. See *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/ #THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

These functions convert the initial portion of the string or wide string pointed to by *nptr* to float and long double representation, respectively. First, they decompose the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the `isspace` function), a subject sequence resembling a floating-point constant or representing an infinity or NaN, and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, they attempt to convert the subject sequence to a floating-point number and return the result.

The expected form of the (initial portion of the) string or wide string is optional leading white space, an optional plus ('+') or minus sign ('-'), and then either (i) a decimal number, or (ii) a hexadecimal number, or (iii) an infinity, or (iv) a NAN (not-a-number).

Returns

The functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus HUGE_VAL, HUGE_VALF, or HUGE_VALL is returned (according to the return type and sign of the value), and the value of the macro ERANGE is stored in *errno*. If the result underflows, the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; whether *errno* acquires the value ERANGE is implementation-defined.

A.1.1.10. va_copy

Format

```
#include <stdarg.h>
void va_copy (va_list dest, va_list src);
```

Description

The `va_copy` macro initializes *dest* as a copy of *src*, as if the `va_start` macro had been applied to *dest* followed by the same sequence of uses of the `va_arg` macro as had previously been used to reach the present state of *src*. Neither the `va_copy` nor `va_start` macro shall be invoked to reinitialize *dest* without an intervening invocation of the `va_end` macro for the same *dest*.

Returns

The `va_copy` macro returns no value.

A.1.1.11. wcstoll, wcstoull

Format

```
#include <wchar.h>
long long int wcstoll (const wchar_t * restrict nptr,
wchar_t ** restrict endptr, int base);
unsigned long long int wcstoull (const wchar_t * restrict nptr,
wchar_t ** restrict endptr, int base);
```

Function Variants

The `wcstoll` function has a variant named `_wcstoll64` for use with 64-bit pointer sizes. The wcstuoll function has a variant named `_wcstoull64` for use with 64-bit pointer sizes. See the *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/#THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

The `wcstoll` and `wcstoull` functions convert the initial portion of the wide string pointed to by *nptr* to long long int and unsigned long long int representation, respectively. First, they decompose the input string into three parts: an initial, possibly empty, sequence of white-space wide characters (as specified by the `iswspace` function), a subject sequence resembling an integer represented in some radix determined by the value of base, and a final wide string of one or more unrecognized wide characters, including the terminating null wide character of the input wide string. Then, they attempt to convert the subject sequence to an integer and return the result.

Returns

The functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, LONG_MIN, LONG_MAX, LLONG_MIN, LLONG_MAX, ULONG_MAX, or ULLONG_MAX is returned (according to the return type sign of the value, if any), and the value of the macro ERANGE is stored in *errno*.

A.1.1.12. Print and scan conversion specifier and argument types

The C RTL now supports the F conversion specifier and the hh, t, j, and z argument types in print and scan.

F	Similar to f.
hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to signed char or unsigned char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.
t	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a *ptrdiff_t* or the corresponding unsigned integer type argument; or that a following n conversion specifier applies to a pointer to a *ptrdiff_t* argument.
J	Specifies that a following d, i, o, u, x, or X conversion specifier applies to an *intmax_t* or *uintmax_t* argument; or that a following n conversion specifier applies to a pointer to an *intmax_t* argument.
z	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a *size_t* or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to *size_t* argument.

A.1.1.13. strftime, wcsftime, strptime – additional conversion specifiers

Description

The following conversion specifiers have been added to `strftime`, `wcsftime`, and `strptime`:

%F	is equivalent to "%Y−%m−%d" (the ISO 8601 date format). [tm_year, tm_mon, tm_mday]
%g	is replaced by the last 2 digits of the week-based year as a decimal number (00−99). [tm_year, tm_wday, tm_yday]
%G	is replaced by the week-based year as a decimal number (e.g., 1997). [tm_year, tm_wday, tm_yday]
%k	The hour (24-hour clock) as a decimal number (range 0 to 23); single digits are preceded by a blank.
%l	The hour (12-hour clock) as a decimal number (range 1 to 12); single digits are preceded by a blank.
%P	Like %p, but in lowercase: "am" or "pm" or a corresponding string for the current locale.
%s	The number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).
%u	The day of the week as a decimal, range 1 to 7, with Monday being 1.

%z	The +*hhmm* or -*hhmm* numeric timezone (that is, the hour and minute offset from UTC).
%Z	The time zone name.
%+	The date and time in date format.

A.2. CRTL ECO V3.0 Changes

For VSI OpenVMS Integrity and VSI OpenVMS Alpha systems, VSI provides the CRTL ECO V3.0 kit that includes bug fixes, new functions, new constants, and a new header file.

For VSI OpenVMS x86-64, all these changes are included in the C RTL.

A.2.1. Bug Fixes

• Calling the function l64a with an invalid argument no longer causes a memory leak.

• Calling the function l64a_r with a null buffer pointer no longer causes an ACCVIO.

• Calling the functions `readv` or `writev` with an invalid file descriptor no longer causes a memory leak.

• Fixed a possible memory leak in `realpath`.

• Fixed possible undefined behavior in `make_cli_comm`.

• Fixed a memory leak in the return path of `newwin`.

• Fixed definitions of `isnan`, `isnanf`, and `isnanl`.

• Fixed `fstat` to return the proper value in the stat field, `st_ino`, when FID$W_SEQ field has the high bit set and _USE_STD_STAT has been defined.

• Fixed headers to define `isblank` and `iswblank` when compiling with `/STANDARD=C99`.

• Fixed the definition of C99 routines when compiling with `/STANDARD=RELAXED`.

• Fixed headers so that nan, nanf, and nanl are only defined when using IEEE floating point.

• Fixed headers so that `va_copy` is only defined when using the latest compiler.

• Fixed SEMAPHORE.H so that it no longer generates a compiler error when compiled with `/STANDARD=ANSI89` or `/STANDARD=VAXC`.

A.2.2. New Constants

The following constants were added to LIMITS.H:

• LLONG_MAX – Maximum value for an object of type long long int.

• LLONG_MIN – Minimum value for an object of type long long int.

• ULLONG_MAX – Maximum value for an object of type unsigned long long int.

A.2.3. New Flags

The following flags were added to DLFCN.H:

- RTLD_GLOBAL

- RTLD_LOCAL

A.2.4. New Datatypes

The following type was added to SOCKET.H:

- socklen_t – Socket address length type.

The following types were added to DECC$TYPES.H:

- typedef const unsigned int * __const_u_int_ptr64;

- typedef int * __int_ptr64;

- typedef const int * __const_int_ptr64;

A.2.5. New Header

This ECO includes MALLOC.H.

A.2.6. Interface Change

The interface for the function `isatty` has been modified.

Previously, in case of an error, the function returned -1. This is not compatible with the POSIX 1003.1 standard. This leads to errors that are hard to find. With this release, in case of an error, the function returns 0 and stores the error in *errno*.

If your code assumes a return value of 0, this means that the fd is not a tty. If your code assumes a return value of -1, this means an error, you will need to change the code. See the following example:

Existing code:

```
int val = isatty(fd);
if (val == 1) {
 // fd is tty
}
else if (val == 0) {
 // fd is not tty
}
else if (val == -1) {
 // error
}
```

Changed code:

```
int val = isatty(fd);
if (val == 1) {
 // fd is tty
}
```

```
else if (val == 0) {
 if (errno) {
  // error
}
else {
  // fd is not tty
}
}
```

A.2.7. Feature Logical Name: DECC$PRN_PRE_BYTE

A change introduced by Hewlett Packard Enterprise (HPE) during OpenVMS V8.4 maintenance allowed systems that used the CIFS product (Samba) to display files in the appropriate format. However, that change affected files with Print File Carriage Control (also known as Fortran Carriage Control). For some environments, the print codes that are removed when transferring files between systems cause incorrect printing behavior resulting in form feeds being lost.

The DECC$PRN_PRE_BYTE feature logical name, when enabled, converts the print codes in files with Print File Carriage Control to their ASCII control code equivalents. CIFS (Samba) then sends them to the client.

Enabling this logical name in addition to enabling DECC$TERM_REC_CRLF, which is used by CIFS, correctly includes the print codes on transferred files.

To enable the DECC$PRN_PRE_BYTE feature, use the following command:

```
$ DEFINE/SYSTEM DECC$PRN_PRE_BYTE ENABLE
```

A.2.8. New Functions

This section describes the functions that have been added to the C RTL. For VSI OpenVMS x86-64, they are included in the C RTL.

For VSI OpenVMS for Integrity and VSI OpenVMS for Alpha, these functions are included in the RTL V3.0 kit.

A.2.8.1. freeifaddrs

Format

```
#include <ifaddrs.h>
void freeifaddrs(struct ifaddrs *ifp);
```

Description

The freeifaddrs function frees the dynamically allocated data returned by the getifaddrs function. *ifp* is the address returned by a previous call to getifaddrs. If *ifp* is a NULL pointer, no action occurs.

A.2.8.2. getgrent_r

Format

```
#include <grp.h>
int getgrent_r(struct group *grp, char *buffer, size_t bufsize, struct
 group **result);
```

Function Variant

The `getgrent_r` function has a variant named `__getgrent_r64` and for use with
64-bit pointers. See the *VSI C Run-Time Library Reference Manual for OpenVMS Systems*
[https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/
#THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

The `getgrent_r` function is the reentrant version of `getgrent`. The `getgrent_r` function
returns a pointer to a structure containing the broken-out fields of a record in the group database. When
first called, `getgrent_r` returns a pointer to a group structure containing the first entry in the group
database. Thereafter, it returns a pointer to the next group structure in the group database, so successive
calls can be used to search the entire database. It updates the group structure pointed to by *grp* and stores
a pointer to that structure at the location pointed to by *result*. Storage referenced by the group structure
is allocated from the memory provided with the *buffer* argument which is *bufsize* characters in size.
The maximum size needed for this buffer can be determined with the `_SC_GETGR_R_SIZE_MAX`
parameter of the `sysconf` function.

If the requested entry is not found or an error is encountered, a NULL pointer is returned at the location
pointed to by *result*.

Returns

On success, the function returns 0, and `*result` is a pointer to the struct group. On error, the function
returns an error value, and `*result` is NULL.

A.2.8.3. gethostbyname_r

Format

```
#include <netdb.h>
int gethostbyname_r(const char *name, struct hostent *ret, char *buffer,
size_t buflen, struct hostent **result, int *h_errnop);
```

Description

The `gethostbyname_r` function is the reentrant version of `gethostbyname`. The caller supplies
a *hostent* structure *ret* which will be filled in on success, and a temporary work buffer *buffer* of size
buflen. After the call, *result* will point to the result on success. In case of an error or if no entry is found,
result will be NULL. The functions return 0 on success and a non-zero error number on failure. In
addition to the errors returned by the non-reentrant version, if *buffer* is too small, the functions will
return ERANGE, and the call should be retried with a larger *buffer*. The global variable *h_errno* is not
modified, but the address of a variable in which to store error numbers is passed in *h_errnop*.

Returns

The functions return 0 on success and a non-zero error number on failure. The global variable *h_errno* is
not modified, but the address of a variable in which to store error numbers is passed in *h_errnop*.

Note

Modules which include calls to `gethostbyname` or `gethostbyname_r` must be compiled with the
C switch `/PREFIX=ALL`.

A.2.8.4. getifaddrs

Format

```
#include <sys/socket.h>
#include <ifaddrs.h>
int getifaddrs(struct ifaddrs **ifap);
```

Function Variants

The getifaddrs function has variants named _getifaddrs32 and _getifaddrs64 for use with 32-bit and 64-bit pointer sizes, respectively. See the *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/#THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

The getifaddrs function creates a linked list of structures describing the network interfaces, one for each network interface on the host machine. The getifaddrs function stores a reference to a linked list of the network interfaces on the local machine in the memory referenced by *ifap*. The list consists of ifaddrs structures, as defined in the include file <ifaddrs.h>. The ifaddrs structure contains the following entries:

```
struct    ifaddrs  *ifa_next;          /* Pointer to next struct */
char               *ifa_name;          /* Interface name */
u_int               ifa_flags;         /* Interface flags */
struct    sockaddr *ifa_addr;          /* Interface address */
struct    sockaddr *ifa_netmask;       /* Interface netmask */
struct    sockaddr *ifa_broadaddr;     /* Interface broadcast address */
struct    sockaddr *ifa_dstaddr;       /* P2P interface destination */
void               *ifa_data;          /* unused */
```

The data returned by getifaddrs is dynamically allocated and should be freed using freeifaddrs when no longer needed.

Returns

The getifaddrs function returns the value 0 if successful. Otherwise, the value -1 is returned and the global variable *errno* is set to indicate an error.

A.2.8.5. getrusage

Format

```
#include <sys/resource.h>
int getrusage(int who, struct rusage *r_usage);
```

Description

The getrusage function provides measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the *who* argument is RUSAGE_SELF, information is returned about resources used by the current process. If the value of the *who* argument is RUSAGE_CHILDREN, information is returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for, the resource information for the child process is discarded and not included in the resource information provided by getrusage.

Currently, only getting elapsed user time (ru_utime) and maximum resident memory (ru_maxrss) is supported.

Returns

Upon successful completion, getrusage returns 0; otherwise, -1 is returned and *errno* set to indicate the error.

A.2.8.6. stpcpy

Format

```
#include <string.h>
char *stpcpy(char *dest, const char *src);
```

Function Variants

The stpcpy function has variants named _stpcpy32 and _stpcpy64 for use with 32-bit and 64-bit pointer sizes, respectively. See the *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/#THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

The function stpcpy uses strlen to determine the length of *src* then copies the *src* to *dest*. The difference from the strcpy function is that stpcpy returns a pointer to the final '\0' and not to the beginning of the line.

Returns

Pointer to the end of the string *dest*.

A.2.8.7. strerror_r

Format

```
#include <string.h>
int strerror_r(int error_code, char *buf, size_t buflen);
```

Description

The strerror_r function is the reentrant version of strerror. The strerror_r function uses the error number in *error_code* to retrieve the appropriate locale dependent error message. The contents of the error message strings are determined by the LC_MESSAGES category of the program's current locale.

If *error_code* is EVMSERR the function looks at vaxc$errno to get the OpenVMS error condition.

Returns

Upon successful completion, strerror_r returns 0 and puts the error message in the character array pointed to by *buf*. The array is *buflen* characters long and should have space for the error message and the terminating null character.

A.2.8.8. strtoimax, strtoumax

Format

```
#include <inttypes.h>
intmax_t strtoimax(const char *nptr, char **endptr, int base);
uintmax_t strtoumax(const char *nptr, char **endptr, int base);
```

Function Variants

The `strtoimax` function has variants named `_strtoimax32` and `_strtoimax64` for use
with 32-bit and 64-bit pointer sizes, respectively. The `strtoumax` function has variants named
`_strtoumax32` and `_strtoumax64` for use with 32-bit and 64-bit pointer sizes, respectively. See
the *VSI C Run-Time Library Reference Manual for OpenVMS Systems* [https://docs.vmssoftware.com/vsi-
c-run-time-library-reference-manual-for-openvms-systems/#THE_64_BIT_SEC] for more information
on using pointer-size-specific functions.

Description

The `strtoimax` and `strtoumax` functions convert strings of ASCII characters pointed to by *nptr*
to the appropriate signed and unsigned numeric values. The `strtoimax` function is a synonym for
`strtoll`, and the `strtoumax` function is a synonym for `strtoull`. These functions recognize
strings in various formats, depending on the value of the *base*. Any leading white-space characters (as
defined by `isspace` in <ctype.h>) in the given string are ignored. The function recognizes an optional
plus or minus sign, then a sequence of digits or letters that may represent an integer constant according to
the value of the base. The first unrecognized character ends the conversion and is pointed to by *endptr*.

Leading zeros after the optional sign are ignored, and 0x or 0X is ignored if the base is 16.

If the base is 0, the sequence of characters is interpreted by the same rules used to interpret an integer
constant: after the optional sign, a leading 0 indicates octal conversion, a leading 0x or 0X indicates
hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

Returns

- If successful, an integer value corresponding to the contents of *nptr* is returned.

- If the converted value falls out of range of the corresponding return type, a range error occurs
  (setting *errno* to ERANGE), and INTMAX_MAX, INTMAX_MIN, UINTMAX_MAX, or 0 is
  returned as appropriate.

- If no conversion can be performed, 0 is returned.

A.2.8.9. strndup

Format

```
#include <string.h>
char *strndup(const char *s, size_t size);
```

Function Variants

The `strndup` function has variants named `_strndup32` and `_strndup64` for use with 32-bit
and 64-bit pointer sizes, respectively. See the *VSI C Run-Time Library Reference Manual for OpenVMS
Systems* [https://docs.vmssoftware.com/vsi-c-run-time-library-reference-manual-for-openvms-systems/
#THE_64_BIT_SEC] for more information on using pointer-size-specific functions.

Description

The `strndup` function duplicates a specific number of bytes from a string. The `strndup` function is
equivalent to the `strdup` function, duplicating the provided string in a new block of memory allocated

as if by using `malloc`, with the exception that `strndup` copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes will be duplicated. If *size* is larger than the length of *s*, all bytes in *s* will be copied into the new memory buffer, including the terminating NUL character. The newly created string will always be properly terminated.

Returns

A pointer to the resulting string or NULL if there is an error.

A.3. Additional C RTL Changes

VSI OpenVMS x86-64 includes the updated C RTL that provides additional C RTL functions, updates to some functions, bug fixes, new header files, a documentation update, and identifies a known limitation.

The current 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

VSI OpenVMS x86-64 Version 9.1-A Field Test and all future versions of VSI OpenVMS for x86-64, including the current release, will contain the C RTL changes implemented in the ECO patch kit RTL V7.0 and later.

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 V8.0.

A.3.1. 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* [https://docs.vmssoftware.com/vsi-c-user-s-guide-for-openvms-systems/ #ALLOCA_SEC] 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* and, if the buffer contains text, store the address of the buffer 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 that 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`. The only difference between `qsort_r` and `qsort` is 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 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 and 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.

A.3.2. 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 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 `writev()`, `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.

A.3.3. 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.

- The `MATH$FP_CLASS_<n>X` functions have been added to STARLET.OLB.

- The `fopen()` and `open()` functions 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 fuction `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 inputio

- The `sem_open()` function no longer fails with the ENOENT error when called with the `oflag` parameter set to 0.

- The lowercase variable name `decc$ga__optarg64` has been added.

A.3.4. 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))
```

A.3.5. 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.

A.3.6. 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, ...);
```