OpenVMS V9.0 x86-64
Boot Manager User Guide

with Virtual Machine Setup

OpenVMS V9.0 x86-64 Boot Manager User Guide: with Virtual Machine Setup

VSI OpenVMS Version 9.0-D for x86-64

Publication date 24-AUG-2020, Document Version 1.5, Document Number: DO-VXBMUG-01A

Legal Notice

The VSI OpenVMS documentation set is available online and on DVD.

Table of Contents

Part I. VSI OpenVMS x86-64 V9.0-D Early Adopters Kit

Getting Started with OpenVMS x86

This User Guide covers a lot of territory. Although it's primary purpose is to describe how to boot OpenVMS x86, a seemingly modest task (type BOOT), it became clear that the subject matter needed to expand in order to achieve this goal when considering the variety of Virtual Machines and installation methods we support.

It is clearly impractical to cover all aspects of using Virtual Machines, so we have to assume that the reader is familiar with their chosen Virtual Machine product and Host operating system.

This guide refers to VSI OpenVMS V9.0 including all incremental dash releases, V9.0-A, -B, -C, -D etc. References to specific versioned filenames, may include a dash designator that differs from the current release. We leave it to the reader to substitute the current dash designator when using this guide.

As always, we welcome any and all comments and suggestions.

Table of Contents

Chapter 1. What's new in this guide for V9.0-D?

Very little has changed in this guide between V9.0-C and V9.0-D. Most notably, Symetric Multi-Processing (SMP) has been enabled in the operating system, so minor updates have been made to describe the enabling of SMP when setting up a Virtual Machine Guest.

Additional changes have been made to the various scripts that compliment this document. The latest scripts can be found on the VSIFTP site in the file titled: *Helpful_scripts.zip*. Each script is accompanied by a brief document explaining the purpose and use of the script. That information supercedes this guide and is not repeated herein.

V9.0-D uses the same V9.0-C Boot Manager utility. You can safely ignore the V9.0-C version message that appears when you launch the Boot Manager.

Chapter 2. About the EAK

VSI OpenVMS Version 9.0-x is an “Early Adopters Kit” (EAK) distributed to a select group of customers having substantial prior experience with the OpenVMS Operating System. The Version 9.0-x Kit is not a complete, production-quality release.

Program development for the EAK is supported through cross-compilers and linkers which run on OpenVMS Itanium systems. Incremental releases will be announced as further native x86 functionality becomes available.

As an early adopter, you have received documentation outlining the content and limitations of the EAK with instructions for obtaining assistance and providing user feedback to VSI.

The EAK is intended to be installed and used as a Virtual Machine Guest OS running under VirtualBox or KVM Host applications.

Note

VSI recommends keeping your VirtualBox or KVM installation up to date. VirtualBox continues to evolve and add more useful features. For reference, at the time of writing, we use VirtualBox Version 6.1. Because we recommend using the latest version, we do not attempt to identify a minimum version.

Host hardware should also be up to date, typically not more than five years old. VSI provides a python script that can be used to verify that your host system has the necessary features to sucessfully run OpenVMS. The same tests performed by the python script are built into the OpenVMS Boot Manager.

Several operating systems can host these Virtual Machine products. Internal to VSI, we have tested under various Apple OS releases, Windows 8, Windows 10, CentOS and other Linux distributions. Because of the variety of host operating systems, it is impractical for VSI to provide detailed instructions for every possible host. We therefore assume that EAK participants are familiar with their chosen host environment and the appropriate Virtual Machine Host application has been installed.

This document provides guidelines for defining a suitable Guest machine configuration from within VirtualBox or KVM.

Note

This guide contains descriptions and screen captures pertaining to VirtualBox. Users of other Virtual Machine Hosts (KVM, etc.) should review the information understanding that the actual procedures will vary on these other Hosts.

Operation on select x86-64 hardware will be introduced in VSI OpenVMS Version 9.1 and later.

Chapter 3. EAK Installation Environment

Note

VSI has chosen to distribute the V9.0-x EAK releases as **pre-configured Virtual Appliances**. Virtual Appliances provide the quickest and easiest way to achieve a ready-to-use system. Each minor release of the EAK will provide a new Appliance with new features that are pre-installed.

The installation methods are described below. Options 2, 3, and 4 will be phased in over future releases.

VSI provides four options for downloading and installing the EAK:

1. Download and Import a Virtual Appliance (.OVA) file.
2. Download a Virtual DVD (.ISO) file (Optical Disk on your Virtual Machine)
3. Create a Physical DVD (Burn an ISO DVD to use with your Virtual Machine)
4. Network boot an installation kit from your own Web Server

Your choice of method depends on your own requirements and how many installations you intend to perform.

The EAK is provided as one or more files available for download from VSI's FTP Server. Access details have been provided to participants in other correspondence from VSI.

Virtual Appliance for VirtualBox: Importing a Virtual Appliance is the fastest way to achieve a fully functional installation. VSI has pre-configured a suitable system, complete with system disk, dump disk, user disk and typical system resource settings. The Virtual Appliance method is intended for the earliest adopters because it does not require running an installation procedure. However, this method does not reflect the official distribution and installation procedures we will use in production environments.

Virtual DVD: Using a Virtual DVD is a simple method if you have a small number of installations, as you can internally distribute copies of the Virtual DVD file. As an ISO-formatted disk image, no further conversions are required. The ISO file can be downloaded and directly attached to your guest as a SATA Optical Disk.

Physical DVD: If you prefer to use a physical DVD, the ISO Kit Disk Image file can be burned to a DVD. In this case, a properly configured Virtual Machine Guest will automatically boot the EAK from your DVD drive. If you choose to burn a DVD, keep in mind that the file is already an ISO Disk Image, so it must be burned as-is to the DVD in order to retain the correct Boot Block and Partition Table structures. Do not attempt further conversions of the file. Many suitable DVD burner utilities are available, for example: Nero Express on a Windows PC works well.

Web Server: The EAK files can be posted to your internal Web Server, allowing it to be accessed and installed by your Virtual Machine Guests. In order to boot the EAK over a network and Web Server, a small UEFI (console) utility named *VMS_KITBOOT.EFI* must be made available to each Virtual Machine Guest. When executed, this utility will prompt for the IP address of your Web Server and download the EAK to the Guest. Details of setting up a Web Server configuration and using VMS_KITBOOT.EFI are provided in later chapters. The VMS_KITBOOT utility is provided as a Virtual Disk in both VMDK and QCOW2 formats to simplify use with a Virtual Machine Guest.

Note

The Web Server boot method is not currently supported for the V9.0-D release.

Each of these installation methods will be described in detail in later chapters.

Part II. Boot Process

A new way of booting OpenVMS

Table of Contents

Chapter 4. Basic Boot

Even OpenVMS Engineers forget how to boot their systems... Here's the short form.

1. If you have not yet defined your Virtual Guest, refer to the appropriate section of this guide to create a suitable configuration. If you have defined your Guest, start it.

2. OpenVMS is booted from the UEFI **Shell>** prompt. If you have configured your Virtual Guest correctly, the UEFI Shell application should start automatically. It is important to use the Shell application that is provided by your chosen Virtual Machine Host. Some Virtual Machine Hosts may require the installation of additional packages to obtain a UEFI Shell. VirtualBox provides a UEFI Shell when the “System Motherboard” option “Enable EFI” is checked. KVM is based on the open source QEMU/OVMF (Open Virtual Machine Foundation) sources which provide the UEFI Shell.

3. Shell> **VMS_BOOTMGR**

4. BOOTMGR> **DEVICE**

5. BOOTMGR> **BOOT {device-name}**

As shown above; from the UEFI Shell> prompt, launch the OpenVMS Boot Manager, select your boot source device and issue the BOOT command specifying your boot device name. If you have previously booted, then your prior boot command will be used by default and you can omit the device name from your BOOT command. Pay attention to your “Default Boot Command” as shown, to be sure you are booting from the desired device. This is especially important after an installation because you will be booting the target device for the first time, yet the previous boot was from the installation media device.

Boot Flags control various aspects of booting. Boot Flags and System Roots can be managed by Boot Manager commands (FLAGS and ROOT) or they can be specified on the initial command line when you launch the Boot Manager. For example, to boot from System Root 0 and hex flag value 807 you could set individual flags from within the Boot Manager or you could issue the command: Shell> **vms_bootmgr dka0: -fl 0,807** (specifying your desired device name). This same command can be inserted into **startup.nsh** if you prefer to always boot the same device. The Boot Manager **AUTO BOOT** command that will provide a brief countdown prior to taking an automatic boot action.

Note

The boot flags for OpenVMS on x86-64 are significantly different from those on prior architectures. Refer to the chapter titled *Boot Modes, Boot Messages and Boot Flags* for details, or simply issue the Boot Manager **FLAGS** command to see the boot flag definitions.

The OpenVMS Boot Manager is independent from the OpenVMS x86-64 Operating System version. Any version of Boot Manager should be able to load any version of x86-64 OpenVMS V9.x. The features offered by specific Boot Manager versions are certain to evolve, but all versions will support essential boot operation. If you want to run a specific instance of the Boot Manager (perhaps to use some new feature), you need to first select the UEFI file system device (FS0:, FS1:, etc) and change directory to where the desired Boot Manager is located before launching VMS_BOOTMGR.EFI. For example: FS1:> cd EFI\BOOT

If you launch the Boot Manager without first selecting the UEFI file system device and directory, platform firmware will scan a search path of devices and launch the first copy of VMS_BOOTMGR.EFI it locates. Often, platform firmware Setup mode allows you to define or constrain this search path so that your desired boot operation is automatic.

Chapter 5. Boot Process Overview

OpenVMS x86-64 Version 9.0 introduces a new boot process. This chapter contains a brief overview of the new process. Following chapters provide greater detail.

Platform Firmware: OpenVMS requires UEFI (Unified Extensible Firmware Interface) console firmware, as opposed to older Legacy BIOS-based consoles.

There is some confusion about the difference between UEFI and BIOS. BIOS (Basic I/O System) is, in general, an older form of platform firmware based in Flash or ROM. BIOS provides the most essential methods for initializing, loading and booting an operating system, but it is a very primitive environment with somewhat inconsistent implementations. In 1982, Intel introduced EFI (Extensible Firmware Interface) which is a loadable set of procedures designed to provide a consistent and feature-rich boot environment. In 2010, EFI was moved to an open-source project (Tianocore) and renamed to UEFI (Unified Extensible Firmware Interface). Practically all systems built in the last decade provide UEFI. UEFI is also typically installed in Flash memory and often as a layer of firmware on top of traditional BIOS. The term "Legacy Boot" refers to booting with traditional BIOS or using a compatibility mode of UEFI which interfaces with BIOS. OpenVMS does NOT support Legacy Boot. The OpenVMS Boot Manager makes extensive use of UEFI's more advanced features.

If a *Secure Boot* feature is present in a platform's UEFI firmware, it must be disabled. OpenVMS provides its own security features. Some commodity systems do not allow disabling of Secure Boot. OpenVMS cannot run on these systems.

Firmware Shell Utility: Upon power-up, x86 systems must be set up to run the UEFI Shell application. The UEFI Shell can be set up to automatically launch the OpenVMS Boot Manager utility or the Boot Manager can be invoked manually from the Shell command line. Being able to reach the Shell> prompt is a fundamental prerequisite for any system to boot OpenVMS. Quite often, the Shell is hidden from users and requires changing some setup parameters. For example, disabling Secure Boot will often enable additional setup menu options for accessing the UEFI Shell.

OpenVMS Boot Manager (new): During an initial installation, the OpenVMS Boot Manager is bundled into the installation media and will be run from the media. After installation, the Boot Manager can be run from any UEFI file system device, as it is not coupled to a specific OpenVMS instance or version. When performing an installation from a Web Server, the Boot Manager is downloaded separately, prior to downloading the full ISO image. After installation, the Boot Manager is typically, but not necessarily, run from the installed system disk.

The OpenVMS Boot Manager should not be confused with a platform firmware Boot Manager (typically blue setup screens). The OpenVMS Boot Manager executes after a system is set up, when we are ready to load OpenVMS. The OpenVMS Boot Manager provides commands for managing boot modes, messages and devices. Potential boot devices are identified and enumerated using OpenVMS device naming conventions. A **BOOT {device-name}** command causes the Boot Manager to locate and load the operating system from the specified boot device.

Memory Disk (new): Unlike prior releases of OpenVMS, all of the files that comprise the core OpenVMS *Kernel* are pre-packaged into a Logical Disk (LD) container file, referred to as the “MemoryDisk”. Whether booting normally or booting an installation kit over a network, the MemoryDisk is loaded into memory by the *Boot Manager* using UEFI physical Block IO drivers. This greatly simplifies and speeds up the boot process and eliminates the need for OpenVMS to provide *Boot Drivers* for every supported type of boot device.

During installation, the MemoryDisk contains the full ISO image, regardless of whether it is downloaded from a Web Server or loaded from a Virtual or Physical DVD. After installation, on subsequent boots, the MemoryDisk contains only the minimal kernel files that are required to achieve the ability to switch to run-time drivers.

Once the Boot Manager loads the MemoryDisk into memory and initiates boot of the “Primary Kernel”, the MemoryDisk remains memory-resident. In the event of a Primary Kernel crash, the same

MemoryDisk is booted a second time into separate memory space, forming what is known as the *Dump Kernel*. The Dump Kernel is a limited-functionality OpenVMS instance which processes a crash dump using runtime drivers and upon completion, initiates a system restart. Because the MemoryDisk is already resident in memory, the Dump Kernel boots nearly instantly and processes the crash data much faster than prior implementations.

Boot Blocks (new): Because the embedded MemoryDisk is a separately bootable entity from the system disk, OpenVMS V9.x supports additional “inner” boot blocks around the MemoryDisk itself. When booting a System disk, the “outer” boot blocks are used to locate these “inner” boot blocks. For the most part, this added complexity is transparent to the user. Related utilities such as BACKUP, PCSI, SYSGEN, etc. have been modified to maintain the integrity and security of the embedded MemoryDisk. The use of *symlinks* allows access to MemoryDisk-resident files from system components.

Chapter 6. Boot Process Detail

If you don't need further details, you can skip this chapter.

The OpenVMS x86-64 boot process is significantly different than the VAX, Alpha or Itanium implementations. The OpenVMS Boot Manager leverages many of the advanced capabilities of UEFI to provide a rich pre-boot environment.

The port of OpenVMS to the x86-64 architecture has also provided an opportunity to modernize and improve the efficiency of the boot process.

The many individual files involved in OS startup are now consolidated into a single *MemoryDisk* image. This offers new options for network booting and greatly enhances boot performance and integrity.

Some of the major features of the Boot Manager and new boot process include:

- A Graphical User Interface, supporting Touch-Pad, Touch-Screen and USB Keyboard/Mouse (on systems that support mouse events).
- A rich command set for customer, support and developer use.
- The Boot Manager is functionally decoupled from the operating system. You can run the Boot Manager from one device and boot a different device.
- Bootable devices are enumerated using OpenVMS device naming conventions. *
- The bootable OpenVMS kernel resides in a single MemoryDisk image file.
- The boot source can be downloaded from a Web Server.
- A second kernel can be initialized to handle crash-dump processing.
- Embedded security features assure boot process integrity.

Note

* Device naming is limited to devices within the scope of UEFI. Some complex storage system devices cannot be enumerated until the OS is running.

MemoryDisk

The most significant change in the boot process is the use of a new “MemoryDisk” boot method.

The minimum bootable OpenVMS Operating System *Kernel* occupies approximately one hundred and sixty files.

On prior architectures, during a network installation, the OS Loader constructed a memory-resident, pseudo system disk containing a set of the essential directories found on a system disk. It then parsed a list of files, downloading and copying each file into its proper directory. The boot process ran from files in this memory-resident disk, eventually reaching a point where it could mount the *real* system disk and use runtime drivers for ongoing file access. The initial memory-resident disk was then dissolved.

This relatively slow and complex process resulted in an OS Loader that needed to understand OpenVMS disk and file structures and was tied to the specific device and version of OpenVMS being booted. It also required special Boot Drivers for every supported boot device. Back when OpenVMS ran on a limited set of systems from a single vendor, this was manageable, but for x86 we expect the OS to run on a much wider variety of systems.

The new *MemoryDisk* boot method eliminates most of this complexity and decouples the OS Loader (Boot Manager) from a specific device or version of x86-64 OpenVMS. Instead of downloading a list of files and constructing a pseudo system disk image in memory at boot time, a single pre-packaged MemoryDisk container file (SYS$MD.DSK) is provided on the distribution media and on every bootable OpenVMS system device.

The MemoryDisk contains all of the files that are required to boot the minimum OpenVMS Kernel. The Boot Manager loads the MemoryDisk into memory and transfers control the primary bootstrap program (SYSBOOT.EXE). It can do this using the physical BlockIO and FileIO drivers provided by UEFI, eliminating the need for boot drivers and knowledge of OpenVMS file systems. All of the files required by the Dump Kernel reside in the MemoryDisk, so the Dump Kernel also directly boots from the MemoryDisk.

As the Boot Manager loads the MemoryDisk, it performs several security and integrity checks before transferring control to SYSBOOT.EXE. Any error in these checks is deemed fatal and results in clearing memory and exiting to the UEFI Shell. Production releases of OpenVMS will use "signed" kernel files, validated by SYSBOOT at load time (not yet implemented in V9.0).

An important concept to keep in mind is that the MemoryDisk is a *container* file that is structured as a bootable system disk, having its own (inner) Boot Blocks and GPT Partition Table, but containing only those files which are required by the early boot process; hence, a “mini-kernel”. When the system is running, the OS uses *symlinks* to direct file references to their actual location if they reside inside the MemoryDisk.

The contents of the MemoryDisk must be kept up to date. Any changes to these MemoryDisk-resident files through parameter file modifications, PCSI Kit or Patch installation, etc., requires the operating system to execute a procedure to update the MemoryDisk container. This assures that the next boot will use the new images. This is handled by system utilities and is transparent to end users. The command procedure: SYS$COMMON:[SYSUPD]SYS$MD.COM handles most of this task.

Note

VSI will be providing documentation for System Managers regarding maintenance of the MemoryDisk, including when and how to run SYS$MD.COM. For V9.0-x, we recommend that you not invoke SYS$MD.COM directly unless you are advised to do so by VSI Support Engineers. If you have customer-developed execlets to load, then you will need to use SYS$MD.COM to recreate the MemoryDisk with your new execlet included. Contact VSI for details. Also, do not rename or move SYS$MD.DSK (the MemoryDisk) or SYS$EFI.SYS (the UEFI partition) as this will invalidate the boot blocks and render the system unbootable.

The MemoryDisk is a Logical Disk (LD) container file named SYS$COMMON:[SYS$LDR]SYS $MD.DSK. Because this is an actual bootable disk image, it implements its own (inner) Boot Blocks consisting of a Protective Master Boot Record (PMBR), a GUID'ed Partition Table (GPT) and multiple disk partition table entries (GPTE's). *A GUID is a “Guaranteed Unique ID” for those who don't recognize the term (similar to a UUID).*

The MemoryDisk image contains two ODS-5 structured partitions (signatures: X86VMS_SYSDISK_LOW and X86VMS_SYSDISK_HIGH) and one FAT-structured UEFI partition (signature: X86_EFI).

An installed *System Disk* also has its own (outer) Boot Blocks consisting of GPT structures and three partitions, similar to the MemoryDisk. The outer boot blocks contain a pointer to the MemoryDisk. Therefore, we can boot a System Disk, extracting the MemoryDisk from it, or we can directly boot a MemoryDisk. Direct boot of a MemoryDisk is primarily a development feature which may be disabled in production releases.

During a normal boot from an installed System Disk, the Boot Manager locates (using the outer boot blocks) and extracts the MemoryDisk from the System Disk. In this case, the MemoryDisk becomes **DMM0:** and is considered to be the system device until the boot process reaches a point where it can

mount the full, physical System Disk. At that point, DMM0: is set offline (but is retained in case the Dump Kernel needs to boot) and the physical System Disk becomes the system device. During the boot process, logical disk device **LDM1:** is created. This is the “logical” MemoryDisk device associated with DMM0, to which symlinks are defined.

Note

Never dismount, disconnect, or remove DMM0: or LDM1:

During a network installation, the entire Installation Kit Disk ISO image is downloaded into memory and (using the outer boot blocks) the embedded MemoryDisk is extracted from the kit. In this one special case and only for the duration of the installation procedure, we have two types of memory-resident disk images; the Installation Kit Disk (DMM1:) and the MemoryDisk contained within the Installation Kit Disk (DMM0:). Both of these memory-resident disks are serviced by the new MemoryDisk driver: **SYS$DMDRIVER.EXE**

UEFI Partition

Of the three OpenVMS disk partitions, the UEFI FAT-structured partition is the only partition that is recognized by UEFI. Thus, during pre-boot execution, UEFI only sees this one partition and the files contained therein. This is where the OpenVMS Boot Manager and any other UEFI executables reside. The other two ODS-5 structured partitions and the files they contain, remain invisible to UEFI.

The UEFI partition is implemented as a Logical Disk (LD) container file named: [SYSEXE]SYS $EFI.SYS. This is a full implementation of the UEFI Partition and the folders and files that are visible from the UEFI Shell including /EFI/BOOT, /EFI/VMS and /EFI/TOOLS. The (inner) Boot Blocks of this Logical Disk contain information for locating the MemoryDisk and then, SYSBOOT.EXE inside the MemoryDisk.

During development, in order to allow the MemoryDisk itself to be a bootable entity, the MemoryDisk container file [SYSEXE]SYS$MD.DSK also contains a reduced-functionality UEFI partition implemented as another Logical Disk (LD) container file named: [SYSEXE]MEM$EFI.SYS. Although there are no UEFI utilities in this *boot-only* UEFI partition, it's (inner) Boot Blocks contain just enough information to locate and load SYSBOOT.EXE when the MemoryDisk is directly booted (this is a development feature).

To be technically complete, an Installation Kit Disk uses an ISO-9660 structure and boot method. Per ISO specification, yet another copy of the UEFI partition exists as the *ISO Boot File*. Per UEFI specification, the ISO Boot File must be a UEFI partition instead of an executable. Therefore, when an ISO DVD is detected by UEFI firmware, UEFI treats the ISO Boot File as a file system partition and if the partition contains a UEFI application named: /EFI/BOOT/BOOTX64.EFI UEFI firmware automatically launches the application. In our case, this is a copy of the OpenVMS Boot Manager. The net result is that the Boot Manager executes automatically when an Installation Kit DVD is inserted in the DVD drive.

Locating the MemoryDisk

The first task for the Boot Manager is to locate the MemoryDisk inside the boot source device regardless of whether we are booting an installed system disk or booting an installation kit over a network.

When booting from a Virtual or Physical device, the boot source device's (outer) Boot Block contains the location (Logical Block Number) and size of the MemoryDisk and the relative location and size of SYSBOOT.EXE within the MemoryDisk. SYSBOOT.EXE is the initial OpenVMS bootstrap program to be loaded and executed.

When booting an Installation Kit Disk over a network the VMS_KITBOOT.EFI utility downloads into memory, a copy of the Boot Manager and the entire Installation Kit Disk image. It then launches

the Boot Manager. The Boot Manager, using the outer boot blocks, locates, extracts, validates and loads the MemoryDisk from within the memory-resident Installation Kit Disk image and then, using the inner boot blocks, locates, extracts, validates, loads and executes SYSBOOT.EXE from within the MemoryDisk.

Before transferring control to SYSBOOT.EXE, the Boot Manager performs a variety of other tasks required to prepare the system for execution including allocation of major data structure memory, enumeration of devices and establishing the memory maps. It does this for both the Primary and Dump Kernels.

If a Bugcheck occurs at any point after OpenVMS is booted, the Dump Kernel, is already resident in system memory (this includes the MemoryDisk and a separate copy of SYSBOOT.EXE). The crashing Primary Kernel passes control to the Dump Kernel, which will rapidly boot, perform its tasks, and reset the system. The use of an independent Dump Kernel provides for faster crash dump processing and allows the use of Runtime Device Drivers (recall that we eliminated the use of Boot Drivers, which were required on prior architectures, to write a dump file).

The operating system procedures that build and maintain the MemoryDisk must also maintain the Boot Blocks of the MemoryDisk and UEFI partition (which are both disk images). If system files that reside within the MemoryDisk are updated, by installation of a patch kit for example, the involved utilities will invoke a procedure to create a new MemoryDisk image so that any changes will be preserved for the next system boot. The system files that reside within the MemoryDisk are not files that a user or system manager would typically need to modify unless you need to add customer-developed execlets to be loaded during boot.

Chapter 7. OpenVMS Boot Manager

The **VSI OpenVMS x86 Boot Manager** loads the OpenVMS operating system on a native x86-64 platform or as a *Guest* operating system under a Virtual Machine Host.

The Boot Manager is a UEFI (console firmware) application located on the boot device, installation media or network distribution site. The actual file name is **VMS_BOOTMGR.EFI**.

Those familiar with booting OpenVMS on Itanium systems will recognize a similar console application named VMS_LOADER.EFI. Although the Boot Manager application is also an *OS Loader* it specifically serves the x86-64 architecture and its functionality differs considerably. The Itanium VMS_LOADER offered very few features. Additional EFI utilities were required to help users identify devices and manage the boot environment. The OpenVMS x86-64 Boot Manager has many built-in features, eliminating the need for additional utilities.

It is important to note that platform firmware often provides its own form of built-in “Boot Manager”, usually presented as a series of simple menus that manage platform setup features and let users define various boot options. Do not confuse these platform-specific setup programs with the OpenVMS Boot Manager. If you are familiar with the Linux environment, you can consider the OpenVMS Boot Manager to be analagous to the *GRUB* boot loader utility, but specifically tailored to the needs of OpenVMS.

To boot OpenVMS, the x86 system must be set up to launch the UEFI Shell application. From the Shell> prompt, we (manually or automatically) launch the OpenVMS Boot Manager. Although the OpenVMS Boot Manager can run without the UEFI Shell, the Shell provides local UEFI file services. Without these services, certain functions of the OpenVMS Boot Manager will not be available. Most notably, the mapping between OpenVMS device names (DKA100, etc.) is not possible and the user will need to boot using UEFI file system device names (fs1: etc.).

PLEASE NOTE: The OpenVMS Boot Manager will evolve to best serve its current users. The appearance, variety of commands, functions and features offered by the Boot Manager will initially serve the VSI development community. Prior to public release the Boot Manager will be refocused to serve the needs of our customers and customer support communities and prior development versions of the Boot Manager will cease to function.

A NOTE ABOUT SECURITY

Operating System Loader applications present a tempting target for hackers. The more sophisticated and interconnected these and other pre-boot applications become, the more enticing they become as targets. The development of UEFI, ACPI, IPMI, SMBIOS and similar pre-boot standards has helped to provide pre-boot compatibility but this has also provided a well documented path for those that look to do harm. Industry attempts to enhance security through firmware, such as the ***Secure Boot*** *initiative and hardware, such as the Intel* ***Trusted Computing Module (TCM)*** *have managed to slow, but not prevent, security breaches. Additional proprietary measures and even dynamic countermeasures may become necessary.*

Prior to production release, the Boot Manager will reduce it's features so as to avoid becoming a useful tool for hackers. The EAK does not yet contain a full regiment of security features. We welcome suggestions for any new features or means of improving security. The problem reporting database is the place to post any problems, suggestions or feature requests.

Part III. EAK Installation

Methods of installing OpenVMS

Table of Contents

Chapter 8. Installing as a VIRTUAL APPLIANCE

Before proceeding with installation, you may wish to verify that your chosen host system has the features necessary to run OpenVMS. A python script has been provided for testing hardware compatibility of host systems. In general, any Intel-based system produced in the last decade should be capable of running an OpenVMS Guest, but some systems may lack one or more critical features. It is always better to know before you start. If you have already installed the Virtual Appliance, you can still verify hardware compatibility using the Boot Manager command: BOOTMGR> DEVICE CPU

Note

This chapter contains descriptions and screen captures pertaining to VirtualBox. Users of other Virtual Machine Hosts (KVM, etc.) should review the information understanding that the actual procedures will vary on these other Hosts.

Scripts for Configuring and Running Guests

In the VSI FTP site, you will find a zip file named **Helpful_Scripts.zip**. The collection of scripts and related documentation can be used to configure, run or remove VirtualBox and KVM virtual machine guests using the pre-configured Virtual Appliance file. These scripts can be modified as needed to fit your environment. These are provided as helpful tools for those who prefer to work with scripts. You are not required to use them.

Scripts Include:

1. How to Configure and Run VSI OpenVMS x86-64 V9.0 in a KVM Virtual Machine on Linux.
2. How to Configure and Run VSI OpenVMS x86-64 V9.0 in an Oracle VirtualBox Virtual Machine on Linux.
3. How to Configure and Run VSI OpenVMS x86-64 V9.0 in an Oracle VirtualBox Virtual Machine on Windows 10.
4. Python script for verifying host hardware compatibility.
5. Other scripts as they are contributed...

What is a Virtual Appliance?

A Virtual Appliance is a Virtual Machine Guest containing a fully configured OpenVMS installation that has been pre-configured and exported. The Guest configuration and disks are compressed and packaged into a single file that can be copied and imported into a different host. This provides an easy way to deploy pre-defined Guests (hence, Appliances). Virtual Appliances are often stored in Cloud Servers.

The “Open Virtualization Format” or OVF is a file type and standard used by some of the more prominent Virtual Machine Host providers. An OVF file describes the Guest configuration. Additionally, a manifest file (.mf) and one file per virtual disk (typically .vmdk) are required to fully represent the Guest. All of these files are then compressed (tar) and packaged into a single appliance file (.OVA) for easy distribution.

Importing a Virtual Appliance into VirtualBox

Note

KVM users should read the VirtualBox details below, then refer to the following section specifically for using our Virtual Appliance on KVM Hosts. Additionally, the scripts provided on the VSI FTP site are helpful for those who prefer to work with scripts.

VSI has provided a Virtual Appliance for EAK customers who would like to be up and running with minimal effort.

The Virtual Appliance that VSI provides contains:

- Physical Memory: 6GB **(Note 1)**
- **Serial Port COM 1 (OPA0): Uses TCP Port 2023 (Note 2)**
- **Serial Port COM 2 (TTA0): Uses TCP Port 2024**
- Network Adapter: EIA0 NAT, Intel PRO/1000 MG Desktop (8250 OEM) **(Note 3)**
- Disk DKA0: 2GB System Disk **(Note 4)**
- Disk DKA100: 2GB Page and Dump File Disk **(Note 5)**
- Disk DKA200: 2GB User Disk (empty)
- USB 1.1 Enabled **(Note 6)**

Note 1: OpenVMS can run in as little as 4GB of memory, but 8GB is the recommended minimum. We have chosen 6GB for this configuration to assure that the Guest will run on minimally configured host systems. If your host has ample memory, you will want to increase your Guest allocation to 8GB or more. If 6GB is not available, you might lower this slightly, understanding that it may impact system performance.
Note 2: These TCP Port Numbers are what you must use to connect a TELNET terminal emulator session to your Guest. The OPA0 port (2023) should be connected when the BOOTMGR> prompt is displayed. The TTA0 port (2024) is optional, but handy once the system is booted. If more than one of the supplied Virtual Appliance Guest are run on the same system, adjust the TCP Port Numbers used for OPA0 and TTA0 to avoid conflicts with the other Guests. Adjust the COM Port settings to avoid conflict too. You are not required to use TELNET and your Serial Ports can be set up to suit your needs.
Note 3: NAT connects to the internet, Bridged connects to your local network.
Note 4: The System disk will contain the current EAK version of OpenVMS. Most of the disks are dynamically sized and will expand at runtime.
Note 5: The Page File is 1GB and the Dump File is 500MB.
Note 6: USB 1.1 is enabled, but the EAK is not yet supporting USB devices. USB 2.0 and USB 3.0 require the *VirtualBox Extension Pack* be installed and licensed.

To use the Virtual Appliance on VirtualBox (using the GUI):

1. Copy the .OVA file from VSI's FTP site to your Virtual Machine Host. Use FTP in binary mode!

2. Rename the file to remove the semicolon and version number.

3. Run VirtualBox.

4. From the VirtualBox *File* menu, select *Import Appliance...*

5. Browse to and select the .OVA file.

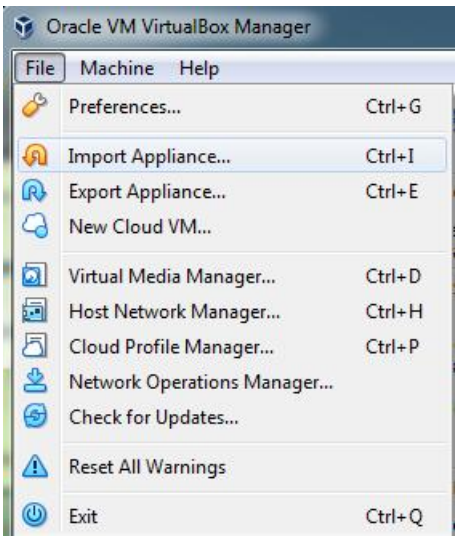
6. Review the Summary.

7. Clear the checkbox labeled *Import hard drives as VDI*

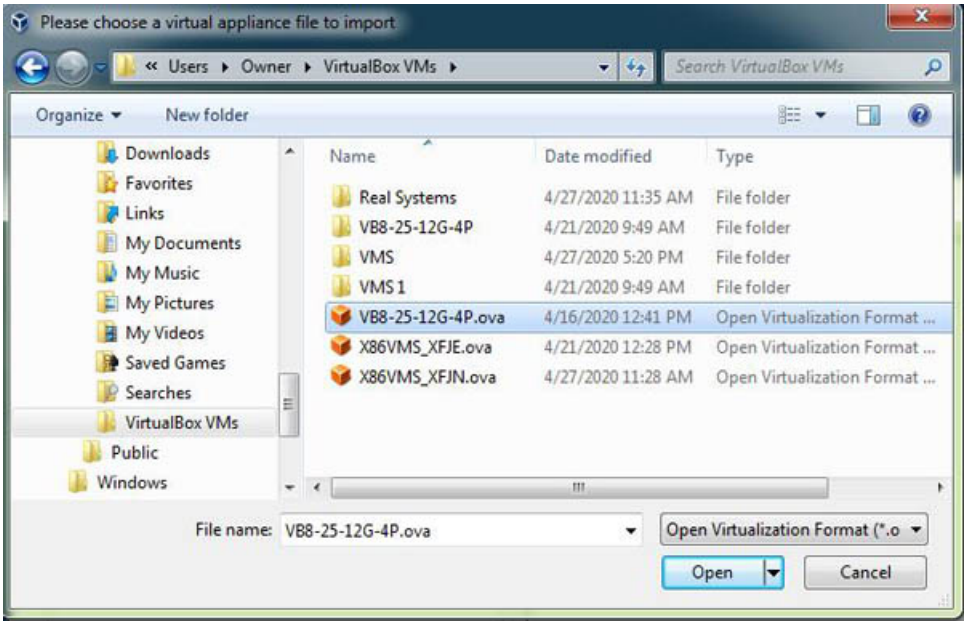
8. Press *Import*

9. Review and resolve any reported configuration issues.

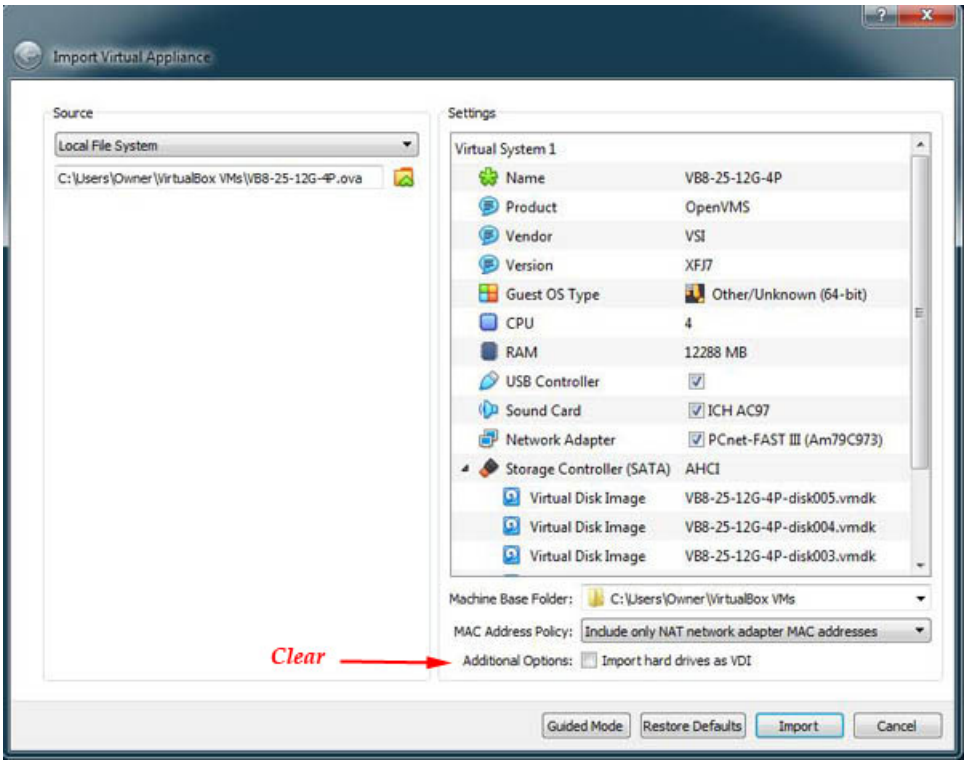
10. Select the new Guest and press *Start*.



Importing a Virtual Appliance



Selecting the Appliance



Virtual Appliance Summary

Note

When Importing an appliance into VirtualBox, you will be presented with a summary of the Guest features. Below this summary is a check box to determine whether to convert the (.VMDK) disks to the native VirtualBox (.VDI) format. Either format works fine, but .VMDK is slightly more portable.

Importing a Virtual Appliance into KVM

Although our Virtual Appliance was produced by VirtualBox, it can be imported into KVM. VSI has provided scripts to illustrate and simplify the required configuration. Refer to the script as a guide, even if you choose not to execute the script.

Note

The scripts provided on the VSI FTP site, **Helpful_scripts.zip>** supercede the information in this guide.

To use an OVA file on Linux/KVM:

1. Copy (FTP binary) the .OVA file to the Linux directory where your VM Guest will be created. Example: */home/KVM/${VMname}/*

   If you used the *make-kvm* script provided in this document to create a Guest and you would like to use that same directory, then your Guest directory will be: */vm_storage/users/${USER}/images/*

2. Untar the OVA file. Example: *# tar xvf vsivmsx86v90D.ova*

   This creates 3 disk files, VSIVMSX86V90D-disk001.vmdk, VSIVMSX86V90D-disk002.vmdk, VSIVMSX86V90D-disk003.vmdk

3. Convert the vmdk files to qcow2 files:

 # qemu-img convert -O qcow2 VSIVMSX86V90D-disk001.vmdk VSIVMSX86V90D-disk001.qcow2

 # qemu-img convert -O qcow2 VSIVMSX86V90D-disk002.vmdk VSIVMSX86V90D-disk002.qcow2

 # qemu-img convert -O qcow2 VSIVMSX86V90D-disk003.vmdk VSIVMSX86V90D-disk003.qcow2

4. Verify for bridged networking that a bridge device has been created called bridge0.

   This device is "easy" to set up by first installing **Cockpit.**

   To install Cockpit:

   a. **# dnf install cockpit cockpit-machines** (Centos 8)

      # yum install cockpit cockpit-machines (Centos 7)

   b. **# systemctl start cockpit.socket**

   c. **# systemctl enable cockpit.socket**

   d. **# systemctl status cockpit.socket**

   e. **# firewall-cmd --add-service=cockpit --permanent**

   f. **# firewall-cmd --reload**

5. Now on your Host, set up a Bridge for your NIC.

   a. Go to https://{ipaddressofsystem}:9090

   b. Select Networking->Add Bridge

   c. Add a bridge named *bridge0* associated with the NIC you want to use.

   d. Press Apply

6. Now create the VM Guest with this script. You can use this as a script or use it line-by-line. Modify it as needed for your VM images location, filenames and number of NICs.

   This script also includes a 1GB transfer disk, consisting of a raw LD container file named: copyfile.dsk. You can create this transfer disk now, or add one later. If you wish to create it now, refer to the section titled *Transferring files between Guest and Itanium systems.*

Note

The scripts in this guide can be cut-and-pasted into an editor, but to be assured of the latest procedures, you can download them from the VSI FTP site (Helpful_scripts.zip) and edit them to suit your needs. Like all things in the Virtual Machine universe, some fine-tuning will be required.

Note

SMP: If you intend to use Symetric Multi-Processing (SMP) you will need to select more than one processor. If you are using the scripts to create your guest, you will need to locate and modify the line which determines how many processors to assign. For example, in the following script you would change *--vcpus 1* to assign more than one CPU. If you are using a Graphical User Interface, locate the equivalent function and increase the number of assigned CPUs.

After assigning additional CPUs, you will need to issue the **SMP** command in the Boot Manager prior to issuing the **BOOT** command. Once enabled, SMP will remain enabled unless it is explicitly disabled via **NOSMP** command.

```
#!/bin/bash
VMname=VMSK1
VMdir=/home/KVM/${VMname}   ...or...  VMdir=/vm_storage/users/${USER}/images/"
declare -i vNet=4
virt-install --connect qemu:///session --name ${VMname} \
--memory 8192 --vcpus 1 --cpu host --hvm --virt-type kvm --arch x86_64 \
--machine pc-q35-rhel7.6.0 --features acpi=on,apic=on,pae=on,vmport=off,smm=on \
--disk ${VMdir}/VSIVMSX86V90D-disk001.qcow2 \
--disk ${VMdir}/VSIVMSX86V90D-disk002.qcow2 \
--disk ${VMdir}/VSIVMSX86V90D-disk003.qcow2 \
--disk ${VMdir}/copyfile.dsk,bus=sata,format=raw,size=1 \
--disk /usr/share/OVMF/UefiShell.iso,device=cdrom,bus=sata \
--check path_in_use=off \
--boot loader=/usr/share/OVMF/OVMF_CODE.secboot.fd,loader_ro=yes,loader_secure=no, \
loader_type=pflash \
--clock tsc_present=yes,hpet_present=yes \
--noautoconsole \
--graphics spice \
--network bridge:bridge0,model=e1000 \
--noreboot

for i in { 2..${vNet} }
do
 virsh attach-interface --domain ${VMname} --type bridge --source bridge0 \
 --model e1000 --config
done

virsh start ${VMname}
```

After successfully executing this script, your VM Guest should be running. There are a few final notes to attend to:

Note

Using the KVM GUI, when you select *Network* and choose a *Network Adapter Type*, the only pulldown selection that shows up as an e1000 is *e1000e*. It actually does have to be e1000, so you have to edit that field to remove the trailing ‘e’.

Note

The KVM GUI only shows 1 NIC. If you need to modify other NICs, you can do so by editing the XML file and locating the specific NICs you need to change or delete, then save. The only way to add additional NICs is by using the VIRSH command shown at the end of the script.

Normally, you would connect to the console terminal on the system via:

virsh console {VM-name}

If you would like to create a second terminal window, you can create a file describing it in XML.

For example, create a file named: **serial1.xml** containing:

```
<serial type='tcp'>
  <source mode='bind' host='127.0.0.1' service='2024' tls='no'/>
  <protocol type='telnet'/>
```

```
  <target port='1'/>
  <alias name='serial1'/>
</serial>
```

Then attach this definition to the KVM VM:

virsh attach-device --config {VM-name} serial1.xml

On the next boot you will have a second terminal that you can connect to via:

telnet 127.0.0.1 2024

Chapter 9. Creating an Installation Target Disk

Note

This chapter contains descriptions and screen captures pertaining to VirtualBox. Users of other Virtual Machine Hosts (KVM, etc.) should review the information understanding that the actual procedures will vary on these other Hosts.

Before performing an installation of OpenVMS from Virtual or Physical DVD or Web Server, you will need to create a Virtual Disk to become your Target System Disk. **The OpenVMS EAK supports only SATA hard disks.** We suggest a minimum of 2GB for a System Disk.

Virtual Disks are typically defined as a fixed-size (flat) or a variable size (sparse). Variable sized disks will dynamically expand as needed to accommodate files, with an optional size limit. This allows the device to occupy only the space required, but performance will suffer if the disk is frequently expanded. A fixed-size disk is slightly faster, but will immediately consume the maximum number of blocks that have been defined.

To create a Target disk:

1. In your Guest *Storage* setup, click on the *Add Hard Disk* icon for your SATA Controller. This will display the *Hard Disk Selector* dialog box.

2. Press the *Create* button to display the *Create Virtual Hard Disk* dialog box.

3. Adjust the slider to set the desired size of your Target disk. We recommend a minimum of 2GB for a System Disk.

4. Select the desired type of disk. We recommend using VDI or VMDK type.

5. Select *Dynamic* or *Fixed* size. Dynamic size will expand as needed up to the maximum size specified. Fixed size will create the full size disk.

6. Press the *Create* button.

7. Your new disk will be listed in the *Not Attached* list. Select the disk and press the *Choose* button. This attaches your new disk to your Guest.

When you create a new disk, the OpenVMS Boot Manager will see the disk as a Block IO device and it will not be given an OpenVMS device name. Only bootable devices are enumerated. However, once the Installation Kit is booted, the installation procedure will recognize the empty disk and list it as a potential Target disk for the installation.

Chapter 10. Installing from a VIRTUAL DVD

> **Note**
>
> For V9.0 EAK users, we recommend using the *Virtual Appliance* installation. The Virtual DVD method described in this chapter is not yet enabled for customer use (VSI has not posted the required files).

> **Note**
>
> This chapter contains descriptions and screen captures pertaining to VirtualBox. Users of other Virtual Machine Hosts (KVM, etc.) should review the information understanding that the actual procedures will vary on these other Hosts.

What is a Virtual DVD?

Whether or not your Guest has a physical DVD reader attached, you can use a DVD image file as though it were a physical DVD. VSI has provided the EAK Installation Kit as an ISO file, suitable for use as a Virtual Optical disk (or for burning to physical DVD media).

One of the benefits of using ISO standard disk images is that the ISO format is recognized by the Virtual Machine Host and no further disk format conversion is required. Simply copy the file to your Guest, attach it as an optical disk and boot!

Using a Virtual DVD image file

We assume here that you have already installed your Virtual Machine Guest and are ready to install OpenVMS. If you have not yet installed your Virtual Machine, refer to the part of this document titled *Virtual Machines*.

1. FTP (in binary mode) the DVD image (.ISO) file from VSI to your Virtual Machine Guest folder.
2. Rename the file to remove the semicolon and version number.
3. Add a SATA Controller. In your Guest *Storage* setup, add a SATA Controller if you don't already have one. Generally, DVD's are given their own controller.
4. Add a SATA Optical Disk.
   a. Click on the round *Add optical drive* icon to bring up an *Optical Disk Selector* dialog box.
   b. Press the *Add* button and select your DVD image ISO file and then press *Open*. This makes the new optical disk available to guests. Note that the file extension (.iso) may need to be in lowercase to be recognized by the Guest.
   c. Select the entry you just added and press *Choose*. The virtual optical disk is now ready for use.
5. When you Start your Guest, the Boot Manager should appear in a few seconds. If not, you can launch the Boot Manager from the UEFI Shell> **BOOTX64 or VMS_BOOTMGR**
6. The Installation Kit should automatically boot. If not, you can manually boot it from the BOOTMGR> **BOOT**
7. Continue with the OpenVMS Installation Menu.

Chapter 11. Installing from a PHYSICAL DVD

Note

For V9.0 EAK users, we recommend using the *Virtual Appliance* installation. The Physical DVD method described in this chapter is not yet enabled for customer use (VSI has not posted the required files).

Note

This chapter contains descriptions and screen captures pertaining to VirtualBox. Users of other Virtual Machine Hosts (KVM, etc.) should review the information understanding that the actual procedures will vary on these other Hosts.

If your Guest has a physical DVD reader attached, you can use a DVD image file to burn a physical DVD for use in your reader. VSI has provided the EAK Installation Kit as an ISO file, suitable for burning to physical DVD media (or for use as a Virtual Optical DVD disk).

Using a Physical DVD image file

We assume here that you have already installed your Virtual Machine Guest and are ready to install OpenVMS. If you have not yet installed your Virtual Machine, refer to the part of this document titled *Virtual Machines*.

1. FTP (in binary mode) the DVD image (.ISO) file from VSI to a local system which contains a DVD burner. Note that the file extension (.iso) may need to be in lowercase to be recognized by the Guest.
2. Rename the file to remove the semicolon and version number.
3. Using a suitable DVD Burner application, burn the file to a DVD. The file is already in ISO form and its structure is critical, so burn the file as-is without any further conversion. At VSI, we use *Nero Express* on a Windows PC to quickly burn the file to a DVD. Many other suitable burner applications exist.
4. Insert your new DVD in the reader that you intend to connect to your Guest system. Some VM Hosts do not recognize a DVD reader until media is inserted.
5. Add a SATA Controller. In your Guest *Storage* setup, add a SATA Controller if you don't already have one. Generally, DVD readers are given their own controller.
6. Add a SATA Optical Drive.
   a. Click on the round *Add optical drive* icon to bring up an *Optical Disk Selector* dialog box.
   b. Press the *Add* button and select your DVD reader and then press *Open*. This makes the new optical drive available to guests.
   c. Select the entry you just added and press *Choose*. The optical disk is now ready for use.
7. When you Start your Guest, the Boot Manager should appear in a few seconds. If not, you can launch the Boot Manager from the UEFI Shell> **BOOTX64 or VMS_BOOTMGR**
8. The Installation Kit should automatically boot. If not, you can manually boot it from the BOOTMGR> **BOOT**
9. Continue with the OpenVMS Installation Menu.

Note

Most Virtual Machine Hosts will recognize and accept ISO-format DVDs, but in some cases the Host OS (notably Windows Hosts) will not recognize the DVD but will still allow it to be accessed by the Guest. If your Guest has a *Secure Boot* mode, this must be turned OFF. Some hosts also require the disk image file to have a lowercase .iso extension name.

Chapter 12. Installing from a WEB SERVER

Note

For V9.0 EAK users, we recommend using the *Virtual Appliance* installation. The Web Server method described in this chapter is not yet enabled for customer use (VSI has not posted the required files).

OpenVMS can be installed from a Web Server. This feature can be useful when you have many installations to do for Guests which may not have DVD readers.

For VSI engineers, this feature has greatly reduced the build/test cycle time. For customers, the feature offers an efficient way to manage multiple OpenVMS Installation Kits for physical systems and Virtual Machine Guests.

Note

For the OpenVMS EAK, we have simplified the Web Server boot method to support booting only the EAK Installation Kit. More advanced menu-based methods will be released with subsequent OpenVMS versions.

Web Server Preparation

Any Web Server can be used to serve installations. Because of the variety of available Web Servers, this document will focus on general setup steps only. We will assume that the Web Server is accessible to your Virtual Machine Host and that your Guest's network configuration is able to access this same network.

1. Establish a **/netboot/** directory on your Web Server. We will assume that you know how to do this on your server.
2. FTP (in binary mode) the following files from the VSI FTP site into your Web Server's /netboot/ directory:
   a. VMS_BOOTMGR.EFI - The Boot Manager
   b. VMS_IMG.BMP - The Boot Manager Background Image (optional but nice to have)
   c. VSIVMSX86V90D.ISO - The OpenVMS V9.0-D (EAK) Installation Kit

Note

If the Host operating system does not recognize OpenVMS file versions, you will need to rename the files to remove the semicolon and version numbers after copying them to your Web Server. Note also that some Web Servers are case-sensitive.

Guest Preparation

We assume here that you have already installed your Virtual Machine Guest and are ready to install OpenVMS. If you have not yet installed your Virtual Machine, refer to the part of this document titled *Virtual Machines*.

Installing OpenVMS from a Web Server presents a *chicken-and-egg* scenario. If nothing is installed on a system, then how do we access a Web Server to download and install an OS? How does the system get onto the network and use HTTP?

UEFI firmware provides HTTP/S protocols to support connection to Web-Servers. The legacy PXE (Pre-eXecution Environment) protocol is being phased out in favor of more capable boot protocols. The most popular of these protocols is known as iPXE. *The 'i' in iPXE has no official meaning, but it is generally thought of as "internet" PXE protocol.* For more information about iPXE, refer to `http://www.ipxe.org`

For the OpenVMS EAK we provide a small UEFI-based Web Server Boot Utility based on iPXE protocol named: **VMS_KITBOOT.EFI**

VMS_KITBOOT.EFI automates most of the Web Server boot process. When launched from the UEFI Shell of your Virtual Machine Guest, the VMS_KITBOOT utility will identify your system hardware, configure a suitable network device, prompt for the IP Address of your Web Server and handle the connection and download protocol for the Installation Kit. When VMS_KITBOOT has finished downloading the kit it will launch the OpenVMS Boot Manager to boot the installation procedure.

Note

Much more sophisticated procedures are possible which provide graphical menus of available kits and other useful commands, but we have chosen to keep installation simple for the V9.0 EAK. If these procedures, in addition to any firewall you might be using, present a security concern, you have the option of installing from a DVD image rather than a Web Server.

When supported, VMS_KITBOOT.EFI will be available on the VSI FTP site in pre-packaged Virtual Disk format.

vms_kitboot.vmdk - VirtualBox Virtual Disk containing the utility.
vms_kitboot.qcow2 - A KVM Virtual Disk containing the utility.
vms_kitboot.efi - The utility by itself. Copy to a UEFI FSx: device or USB Stick.

Note

If you choose to use the .efi version, as opposed to a pre-packaged virtual disk, you must create a USB stick containing VMS_KITBOOT.EFI on a system that supports the FAT32 file system (such as a Windows PC) as this is the file system expected by UEFI. Or if your Guest already has access to another EFI partition, you can put this file in that partition. We find it easier to use the virtual disk.

1. FTP (in binary mode) the appropriate version of VMS_KITBOOT.xxx to your Guest folder. Each Guest will require its own copy of VMS_KITBOOT, or you can share a hard drive containing the utility among multiple Guests if they have a common Host.

2. Add a SATA or IDE Controller. The disk image can be attached as an IDE or SATA Hard Disk. If you attach it as a SATA disk, it will appear as a DK device under OpenVMS after installation, but it's contents cannot be directly accessed by a user due to its use of a FAT partition. If instead, you attach the device as an IDE device, it will still be accessible to the UEFI Shell but OpenVMS will not list it (OpenVMS does not currently support IDE disks once booted). For this example, we will use IDE.

3. Select the controller and press the *Add Hard Disk* icon.

4. Add an IDE Hard Drive.

   a. Click on the round *Add hard drive* icon to bring up a *Hard Disk Selector* dialog box.

   b. Press the *Add* button and select your disk image file and then press *Open*. This makes the new drive available to guests.

   c. Select the entry you just added and press *Choose*. The disk is now ready for use.

5. When you Start your Guest and reach the UEFI Shell, launch VMS_KITBOOT using the command: Shell> **vms_kitboot**.

   If you are unable to launch VMS_KITBOOT, refer to the chapter describing your chosen Virtual Machine and review the setup instructions. If you still cannot launch VMS_KITBOOT, contact VSI Support for assistance. Keep in mind that VMS_KITBOOT is expecting your Web Server to be accessible and have the required files located in the **/netboot/** directory. If you change the location or file names, VMS_KITBOOT will fail.

6. VMS_KITBOOT will prompt you for the IPV4 (xx.xx.xx.xx) address of your Web Server. You can also use a named address (mywebserver.mycompany.com), but the use of numeric address will avoid a DNS lookup. The Web Server should respond by downloading the set of installation images.

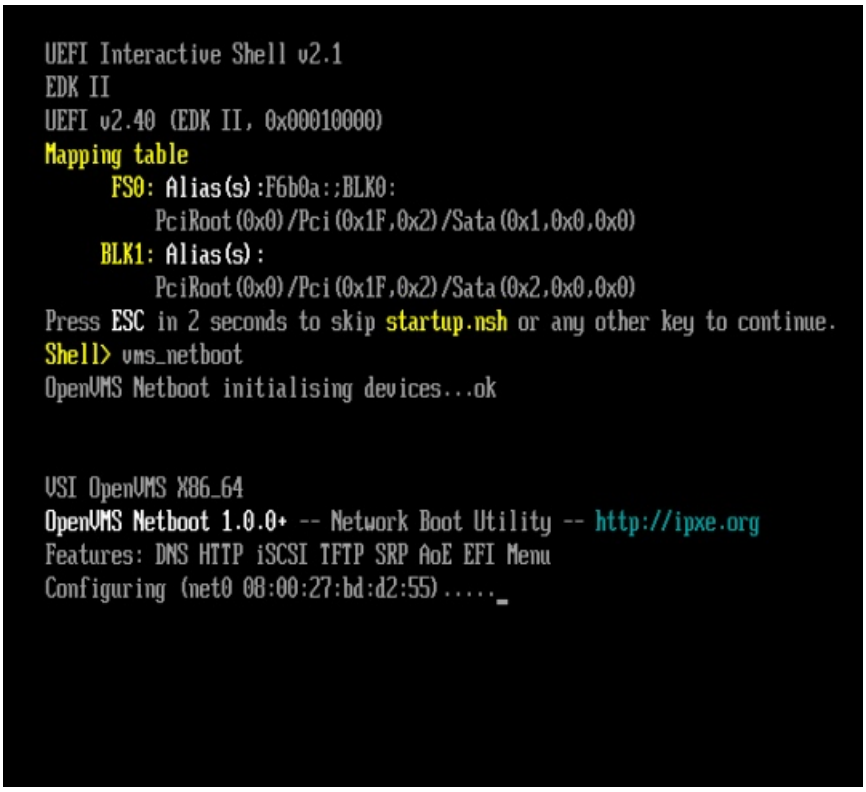
   For an initial Web Server based installation, the downloaded file is the full ISO Installation Kit Disk. This large file can take several minutes to download depending on the speed of your local network. Once the kit has been downloaded, the installation proceeds much faster than it would if each file were read individually.

   Note that this does create a greater, but not unreasonable, demand for available Guest memory during an installation. The kit image occupies approximately 1GB of memory during installation.

   If the download fails due to lack of memory, return to your Guest setup and increase the amount of memory allocated to the Guest. Once the installation completes, you can reduce the memory if you like.

7. When the download completes, the Boot Manager should appear and automatically initiate a boot of the Installation Kit. Note that the memory-resident Installation Kit disk image will appear as DMM1: to OpenVMS during the installation.

8. Continue with the OpenVMS Installation Menu.



UEFI Shell and VMS_KITBOOT Launch Screen

Chapter 13. Completing the Installation

Terminal Connection

The Boot Manager provides a bridge between the UEFI pre-boot environment and the early phase of system boot (SYSBOOT). Once control is transferred to SYSBOOT, the Boot Manager keyboard is no longer available for input and the Boot Manager display shows only XDELTA and Operator terminal output. Normal interaction with OpenVMS is through terminal session connection (TTA0: etc.) for both keyboard input and display output.

A variety of terminal utilities can be used with OpenVMS. Telnet, SSH, Pipes and TCP are the most commonly used methods. It would be impractical to document every known terminal utility, so this document focuses on Telnet and assumes that the user can translate the methods to their chosen terminal utility.

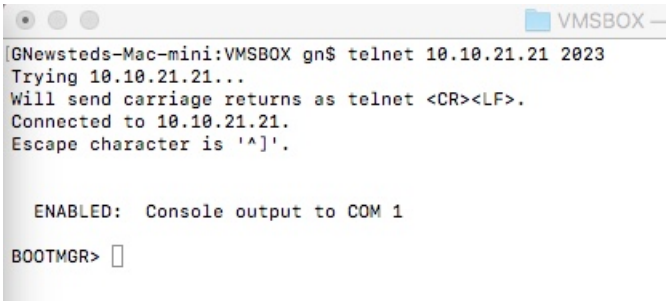
As a general guideline, OpenVMS assumes control over command processing, so you should ENABLE *character* command mode as opposed to *line* mode and DISABLE any functions that would modify the command line terminators (Carriage Returns and Line Feeds). Terminal utilites that attempt to negotiate connection details should be set up to avoid doing so. For example, Telnet may attempt auto-negotiation whereas a *Raw* connection won't. The result is that you may need to adjust your Telnet session to avoid duplicate line feeds, although a Raw session should work without similar concern.

Named Pipes are also an acceptable way to connect to the Boot Manager. In the VBOX serial port setup and in your PuTTY or compatible terminal utility, define your named pipe as: \\.*pipe**COM1*

When the BOOTMGR> prompt appears, connect your terminal emulator to your Guest. If the Boot Manager doesn't automatically sense the connection, issuing the command: BOOTMGR> **COM 1** will initiate command output to the serial port. If you don't establish a connection, OpenVMS will still respond to new terminal connections once booted. You will only miss the Boot Manager messages during boot.

The Boot Manager and UEFI use 115200 baud, 8 bit, 1 stop bit, no parity. It will auto-negotiage other baud rates.

Refer to the Troubleshooting chapter if you have problems establishing a terminal session.



Terminal Session after Connecting

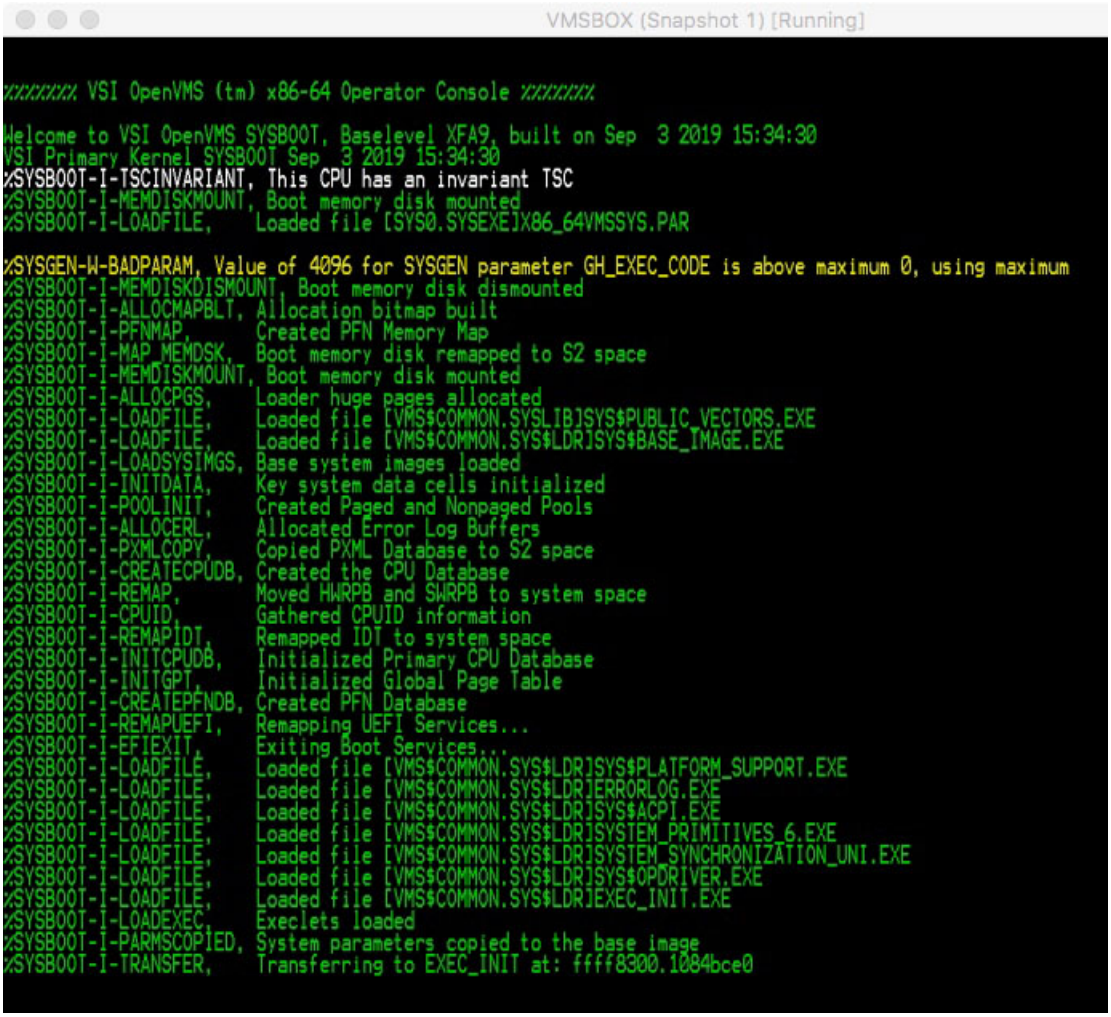
Booting the Installation

We recommend setting the *PROGRESS* message boot flag before proceeding. To set this flag, enter the BOOTMGR> **PROGRESS** command. Additional flags will produce more information during the various stages of the boot process.

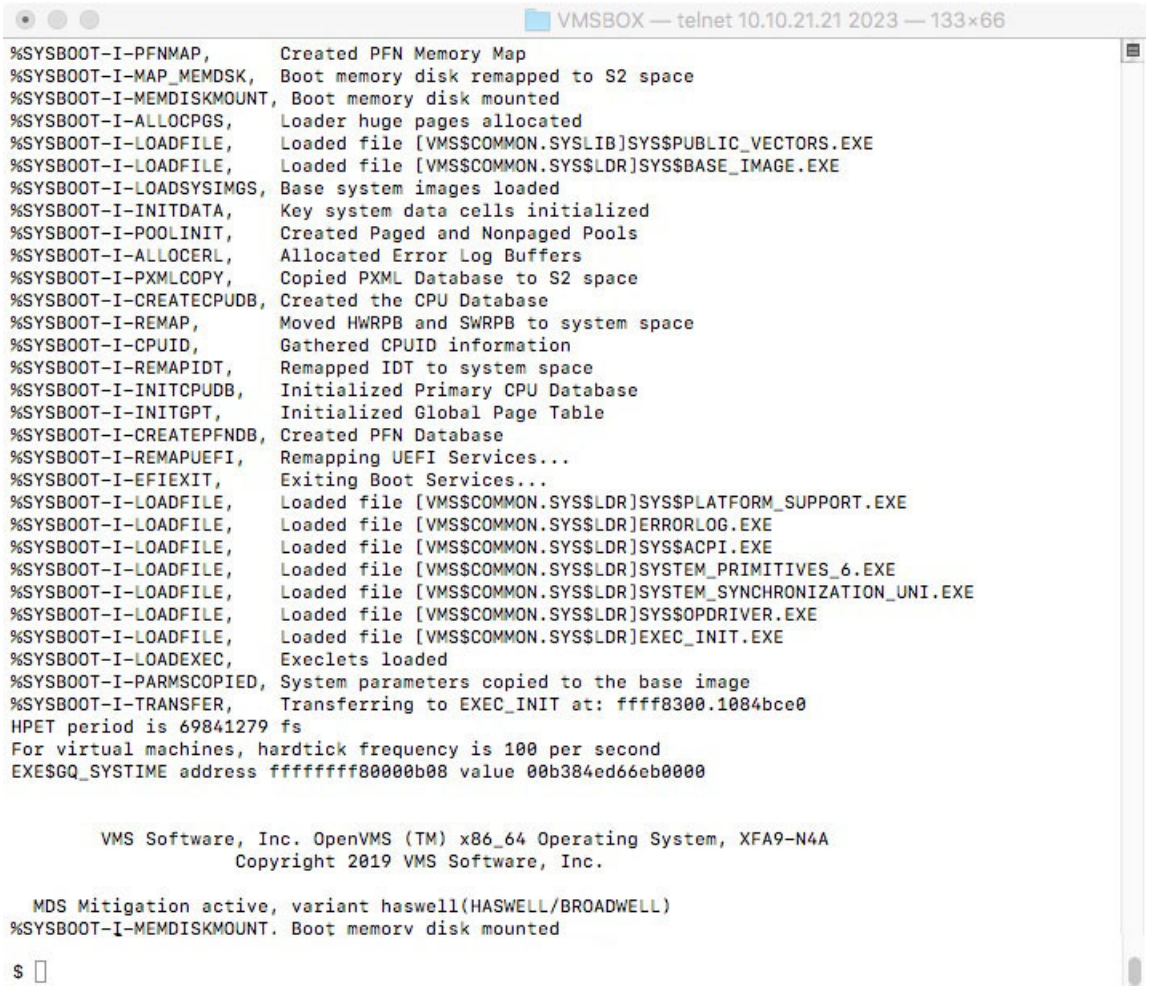
To initiate the installation boot, enter the BOOTMGR> **BOOT** command. For an installation, it is not typically necessary to specify a boot device because the installation device will already be the default device.

If you are performing a Web Server installation, the default device will always be DMM0.

If you are unsure of the device, use the BOOTMGR> **DEVICE** command to list the available boot devices and device types. Note that individual disk partitions will have UEFI FSx and BLKx device names. Pay attention to those which are listed as bootable VMS devices.



Progress Messages on Guest Display after BOOT



Progress Messages on Terminal Screen after BOOT

OpenVMS Installation Menu and Post Installation Steps

As the boot proceeds, you will eventually reach the OpenVMS Installation Menu. At this point, the installation procedures are similar to a traditional installation. You will be prompted to select your target device to install to and any optional kits you wish to install.

Note

Refer to a prior version of the VSI OpenVMS Installation Manual for further details. Note that the Installation Manual has not been updated for the V9.0 EAK, but the installations are similar to prior versions from this point.

When the installation completes, the Guest will be reset. Upon reset, your Guest should return to the UEFI Shell> prompt. At this prompt, perform a *normal* system boot as follows:

1. Shell> **VMS_BOOTMGR**

   When the BOOTMGR> prompt appears, connect your terminal emulator session.

2. BOOTMGR> **BOOT** *target-device-name*

It is important that you specify the Target device where you installed OpenVMS on this first subsequent boot. If you wish to have the system automatically boot, issue the command BOOTMGR> **AUTO**

BOOT before booting. This will preserve the next successful boot device as the default. Subsequent boots will occur after a brief countdown whenever the Boot Manager is launched.

To cause the Boot Manager to be launched automatically you can do one of two things:

On the UEFI File System device representing the disk you installed OpenVMS on, in the */EFI/BOOT* folder:

1. Define a Boot Option to specifically launch VMS_BOOTMGR.EFI

2. or

3. Edit startup.nsh and insert the line: VMS_BOOTMGR.EFI

If you choose to use startup.nsh, you can also specify a fully qualified boot command such as: **vms_bootmgr DKA100: -fl 0,0** where the -fl arguments specify a System Root and Boot Flags.

Note

If you edit startup.nsh on an OpenVMS system, you will need to convert the file back to stream-lf format before copying back to your UEFI partition. The way to do this is: $ **SET FILE/ ATTR=RFM:STMLF STARTUP.NSH**

Now when your Guest starts the UEFI Shell, the firmware will recognize one of these methods and launch the Boot Manager. The Boot Manager will execute its countdown and if not interrupted by user input, automatically issue the boot command.

This concludes the information in this document pertaining to installation booting. If you need more information to complete your installation contact VSI Support.

Part IV. Boot Manager

An OS Loader and more...

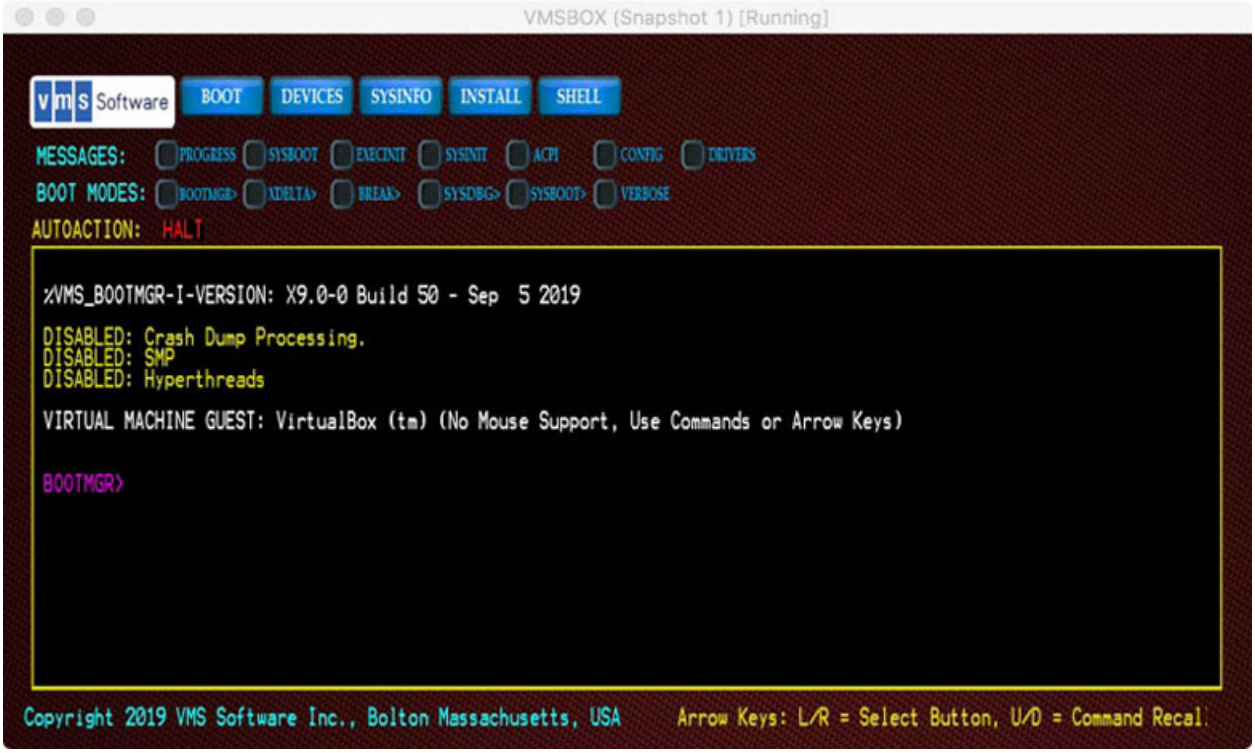
Table of Contents

Chapter 14. Boot Manager Basics

This chapter describes the operation of the OpenVMS Boot Manager. Most customers will seldom need more than the basic features of the Boot Manager. Normally functioning systems can be set up to automatically boot with no interaction with the Boot Manager at all. The more advanced features of the Boot Manager provide functions required to initially establish the desired boot environment, set it up for automatic booting and diagnose boot problems.

Prior to production release, VSI will evaluate which features are appropriate for the customer environment and disable any developer-focused features.

Display and Input



The Boot Manager Startup Screen (in Graphical Mode)

The Boot Manager attempts to operate in the most sophisticated manner that is supported by the system being booted. There are four distinct modes of Boot Manager operation, depending on the capabilities of a system and its firmware.

The Boot Manager evaluates and selects the most capable mode in the following order:

- **GRAPHICAL_IO Mode:** The system provides full graphical display features and a mouse, touch-pad, or touch-screen input device. In this mode the Boot Manager supports push buttons, check buttons and a graphical background image.
- **GRAPHICAL Mode:** The system provides full graphical display features but *does not* provide a mouse, touch-pad, or touch-screen input device. In this mode the Boot Manager supports push buttons, check buttons and a graphical background image, but the keyboard arrow keys must be used to interact with these graphical controls.
- **RENDERED_TEXT Mode:** The system supports simple graphical display features such as full screen colored text but without graphical controls or a background image. Input is limited to keyboard only. On systems that normally support the graphical modes, Rendered_Text mode

can be intentionally selected by deleting or renaming the background image file (fsn:\EFI\BOOT\VMS_IMG.BMP).

- **SERIAL_TEXT Mode:** The system does not support graphical display features. Input and output will be routed to a physical or virtual serial communications port.

Output can be routed to one or more serial ports even when operating in one of the graphical modes. This can be useful for logging console output to a terminal emulator. The Boot Manager itself, does not provide logging, although it has a screen capture feature.

Custom Background Image

On systems that provide a graphical display, the Boot Manager will attempt to load an optional background image file that may be customized by the user. A default background image has been provided by VSI.

The background image file must adhere to certain guidelines pertaining to format, dimensions, color depth, location and name.

The file location and name must be: **fsn:\EFI\BOOT\VMS_IMG.BMP**. If this file is found and it meets the specified guidelines, it will be used by the Boot Manager.

The background bitmap image must be non-compressed, BMP format with 24-bit color depth. Image dimensions must be a minimum of 1024 x 768 pixels and ideally, 1920 x 1080 pixels to support most common HD resolution monitors. If the image is smaller than the current display, it will be tiled as needed to fill the display. If the image is larger than the current display, it will be cropped. Image scaling and rotation are not supported.

A region in the upper left corner of the image, measuring 900 x 144 pixels, is reserved for overlaying the various buttons and the VSI Logo. The user cannot alter the placement or color of the graphical controls, so any custom image should provide adequate contrast in this region to assure that the controls remain visible. Selected controls are highlighted in red, so this too should be considered when selecting a custom image (avoid red). A VSI Copyright statement will be overlayed along the lower edge of the image. The text output area will be overlayed on top of the provided image.

If no suitable background image is found, the Boot Manager will fall back to RENDERED_TEXT mode. In this mode the full display dimensions will be used for output. If this full size display mode is desirable, the user may rename or delete the bitmap image file. This effectively prevents the Boot Manager from entering GRAPHICAL modes.

For security reasons, the graphical subsystem's frame buffer memory is cleared prior to transferring control to the operating system.

Display within a Parent Window

Some remote server management utilities and Virtual Machine Hosts will run the Boot Manager within a child window that is smaller than the physical monitor. Without having specific interfaces to these various utilities, the Boot Manager is unaware of the size of its containing window and will attempt to size its display according to the physical monitor size. This results in the need to scroll the window to see the complete Boot Manager display.

To improve usability when confined to a child window, do the following:

1. If you are using VirtualBox, the Host provides a top menu item (when running) to scale the Guest display. Unfortunately, not all Hosts provide this feature.

2. Alternatively, the BOOTMGR> **LINES** command, will display line numbers along the left side of the text output region.

3. Determine how many lines are visible without having to scroll.

4. Issue the command again with the desired number of lines. For example: BOOTMGR> **LINES 34**

This will cause the Boot Manager to recalculate its display height to fit the window. This can also be handy if you would like to shrink the Boot Manager to save screen space. Changes to the display size are preserved in response to a BOOT, EXIT or SAVE command. If you wish to restore the full display size, issue the LINES command specifying an excessively large number of lines and the Boot Manager will determine the maximum number of lines it can use with the current monitor.

Note that the Boot Manager does not support display *width* adjustment using the LINES command.

Page and Line Scrolling

The Boot Manager does not support the conventional line-by-line method of scrolling text output.

When booting with additional messages enabled, the amount of text displayed can span many pages. In this case you may issue the **PAGE** command to enter "Page Mode". This will cause the display to pause at the end of each page and wait for the user to press a key to continue. The display will be cleared as each subsequent page is displayed.

Commands that produce more than a single page of output will automatically pause as each full page has been displayed.

Page Mode greatly improves the display performance when using remote server management utilities, particularly when the display is being sent to a Web Browser interface. You can disable Page Mode using the **NOPAGE** command. Page Mode state is preserved across boots.

Routing and Logging Boot Manager Output

Occasionally it is useful to capture boot process output for presentation or problem reporting. End-to-end logging of the boot process can be difficult to accomplish due to the many transitions that occur during the boot process. For example, the graphical or text mode Boot Manager runs within UEFI “Boot” context. It transitions into UEFI “Runtime” context as it transfers control to OpenVMS. In UEFI Runtime context and beyond, the Boot Manager has no access to UEFI file protocols, thus precluding UEFI file based logging. When OpenVMS is able to initialize its Operator Terminal Driver it can commence logging to an OpenVMS file. Seamless logging of output across these complex transitions is best accomplished by routing Boot Manager and Operator output to a terminal session where logging of session output is possible.

When booting OpenVMS as a Virtual Machine Guest, the Virtual Machine Host will often provide a means of logging a session. You may also define a serial port for your Virtual Machine that can be directed to a TCPIP port and address. In this case, you can route the output to a terminal emulator to provide both scrolling and logging capability.

The Boot Manager **COM x** command, where 'x' is the x86 architecturally defined COM Port number (1 through 4), will cause Boot Manager output to be echoed to the specified serial port. **COM 0** disables this function. You may route output to multiple ports by issuing additional COM x commands.

COM 1 (0x3F8) will always become the default port for OPA0 terminal communications once OpenVMS has booted, even if Boot Manager output was disabled via COM 0 command.

Your selected COM port routing is preserved when you issue a BOOT, EXIT, or SAVE command.

Capturing and Projecting Graphical Display Output

A useful Boot Manager feature is graphical screen capture. Pressing function key **F1** at the BOOTMGR> prompt or at any PAGE break prompt, will cause the Boot Manager to capture the current

screen to a UEFI file: **fsn: snapshot.bmp**. *Note: Currently, only a single snapshot file is supported. Subsequent captures will overwrite the prior capture unless you copy or rename the previous file. Be careful to avoid filling up the UEFI partition with these large bitmap files.*

Occasionally it is useful to project the display to a larger screen for demonstration or training purposes. This can be problematic due to the number of display resolution changes that occur between power-up BIOS, UEFI, Boot Manager and OpenVMS. Experience has shown that most modern projectors will synchronize to the resolution used by UEFI. This is fine for projecting the initial power-up sequence. Once you launch the Boot Manager, the resolution will almost certainly increase and some projectors are not able to recognize the change. To accommodate this, we recommend that you avoid enabling output to your projector until you have launched the Boot Manager. This allows the projector to synchronize to the active resolution used by the Boot Manager.

Input Devices

Note

Many Virtual Machine Hosts fail to provide mouse movement notifications while executing in UEFI and Boot Manager context. Keyboard input, initially using UEFI protocols, will be disabled during transition between UEFI and the operating system. Once the operating system is running, it will establish terminal devices as needed. Because OpenVMS V9.0-x does not yet provide USB Keyboard drivers, your locally attached keyboard may not be usable once booted. Instead, user input is expected to occur through a terminal utility.

User input on x86 systems is typically provided by a locally attached keyboard. This may be a standard keyboard, a USB keyboard, a local or remote serial terminal or terminal emulator. Some systems require that the console keyboard be plugged into a specific port in order to recognize it as a boot keyboard. This is particularly true for USB keyboards on systems with firmware that doesn't yet support USB 3.x speeds. Refer to your specific system documentation.

Many x86 systems, particularly in the mid-range and low-end space, do not provide traditional (RS232/454) serial ports. Hardware design requirements, driven by certain retail/entertainment operating systems, forbid the use of these serial ports due to incompatibilities with plug-and-play strategies. The fear is that devices can be connected and configured, then removed or exchanged with other devices without notification to the operating system. For this reason, traditional serial ports have been replaced by USB ports as the de-facto standard for console input devices.

USB input devices have their own set of issues. USB operation requires an active System Timer interrupt and more system resources to be available than required by a simple RS232 serial port. Developers that need to work in the early boot stages should use traditional serial port input in order to avoid temporary loss of console input during boot.

As the system boots, the various input devices must transition to a runtime driver environment. During this transition, there may be a brief period of time (typically a second or two) when these devices become unavailable. For developers using the XDelta debugger, this problem is avoided by using a traditional serial port keyboard or special serial debug port device instead of the USB keyboard.

The Boot Manager and system firmware support a minimum subset of keyboard functionality as required to boot the system. Special function keys and languages other than English are not currently supported by the Boot Manager. Systems that provide a mouse, touch-pad or touch-screen input device should function as expected, but special features of these devices, such as gesture recognition or specialized buttons, are not supported in the boot environment. Some Virtual Machine Hosts, such as VirtualBox, do not report mouse movement at all in the UEFI environment.

The keyboard UP and DOWN Arrows operate a command recall buffer. The RIGHT and LEFT Arrow keys move input focus to the next or previous button in a circular chain of buttons. Whenever the Arrow keys highlight a button, the ESCAPE or ENTER keys will activate the highlighted button. If the user presses any key other than the Arrow keys, input focus is returned to the Boot Manager command line.

Environment Variables

OpenVMS has traditionally been dependent on UEFI Environment Variables to control certain aspects of the boot procedures. For x86 OpenVMS, more emphasis is being given to running as a Virtual Machine Guest operating system. We have noted an inconsistency among the behavior of several Virtual Machine Hosts with regards to how they handle UEFI Environment Variables.

While most Virtual Machine Host applications support interfaces for managing Environment Variables, several fail to provide persistence of these values across power cycles. Worse yet, some Virtual Machine Hosts fail to provide the fundamental interfaces and also fail to report errors when these standard UEFI features are not available.

In order to address this problem, the Boot Manager maintains an internal structure containing the required variables and it stores or retrieves these variables from a UEFI binary file: **fsn:\EFI\BOOT\VMS_ENV.DAT**

Using a binary file to store environment variables assures that they will work with the Virtual Machine Hosts that we have tested. The VMS_ENV.DAT file should not be edited. Nothing in the file is critical and the file is deleted if it fails various validity checks.

Chapter 15. Boot Modes, Boot Messages and Boot Flags

Boot Modes

There are many optional controls that affect the OpenVMS boot procedures. Control of Boot Modes and Boot Messages can be accomplished using Boot Flags. However, most commonly used features also have Boot Manager commands or buttons that are easier to use than flags. This chapter describes these modes, messages and flags, and how they affect the boot procedures.

BOOT MODES:

- **AUTO-ACTION:** Most users will chose to set up their systems to boot with little or no interaction. Systems set up to auto-boot are assured the fastest recovery following a change in power state or fatal system event. The Boot Manager command: **[NO]AUTO** enables or disables a brief countdown before executing the previous default boot command.
- **BOOTMGR INTERACTION:** The Boot Manager can operate in an interactive mode that is designed to help diagnose boot problems. The Boot Manager command: **[NO]BOOTMGR** enables or disables the interactive features that provide insight into the earliest UEFI phase of boot.
- **XDELTA:** Developers can use the XDelta instruction level debugger in one of two ways. To debug the early boot process, XDelta is linked into SYSBOOT. To debug later phases of operation, XDelta will be loaded by SYSBOOT as a loadable (execlet) image. The Boot Manager command: **[NO]XDELTA** enables the XDelta debugger that is linked into SYSBOOT. This is most useful to developers working in the early phase of boot where the system is still using physical addressing.
- **BREAK:** When used along with the XDELTA or SYSDBG modes, the instruction level debugger will respond to the first breakpoint it encounters, typically a call to **ini$brk()** in a code module.
- **SYSDBG:** Developers can use the XDelta instruction level debugger in one of two ways. To debug the early boot process, XDelta is linked into SYSBOOT. To debug later phases of operation, XDelta will be loaded by SYSBOOT as a loadable (execlet) image. The Boot Manager command: **[NO]SYSDBG** enables the loadable XDelta execlet debugger. This is most useful to developers working in later phases of boot where the system is using virtual addressing. It is also useful to device driver developers and for other system components.
- **SYSBOOT INTERACTION:** The Boot Manager command: **[NO]CONVERSATION** enables or disables a pause in the boot process at the **SYSBOOT>** prompt. This is also known as a “Conversational Boot”. A plethora of options are available at this prompt. Consult the appropriate documentation for greater details.
- **VERBOSE:** This mode works in conjunction with the various message-related boot flags. When in VERBOSE mode, OpenVMS System components may produce extended diagnostic information of interest to developers. This extended information is subject to change and does not follow any message formatting rules. Used with certain other flags, the amount of diagnostic data is substantial and may delay booting by many minutes. It is meant to be used by VSI Engineers and VSI Support personnel.

Boot Messages

Developers and Support Personnel may benefit from detailed messages during boot procedures. The source and verbosity of these messages are controlled by Boot Flags. However, most commonly used message controls have Boot Manager commands or buttons that are easier to use than hexadecimal flag values.

In an effort to manage the number of messages that OpenVMS can produce during boot, OpenVMS on x86-64 has defined a subset of Boot Flags to control messages from specific phases of the boot procedures. The Boot Manager provides commands and buttons to control each of these phase-specific messaging features.

The boot phases having their own boot message flags include: BOOTMANAGER, SYSBOOT, EXECINIT, SYSINIT, ACPI, CONFIG and DRIVER. Both graphical mode check buttons and commands are provided for each of these flags.

In addition to each phase-specific message flag, the command or button **[NO]VERBOSE** controls the extent of message detail.

If the VERBOSE mode flag is NOT set, the phase-specific messages will be properly formatted OpenVMS messages of the form: "%FACILITY-Severity-Mnemonic, Message String", for example: *%SYSBOOT-I-PARAM, Loaded Parameter File.* These messages are intended to provide basic progress indication and highlight interesting operations.

If the VERBOSE mode flag is SET, extended debug information will be provided. These messages are not held to format requirements and will take whatever form the developers felt would be useful for detailed troubleshooting.

Boot Flags

Boot Flags are a 32-bit mask of numeric constants that enable or disable specific boot related features. These features include Boot Modes, Boot Messages and a variety of less common features.

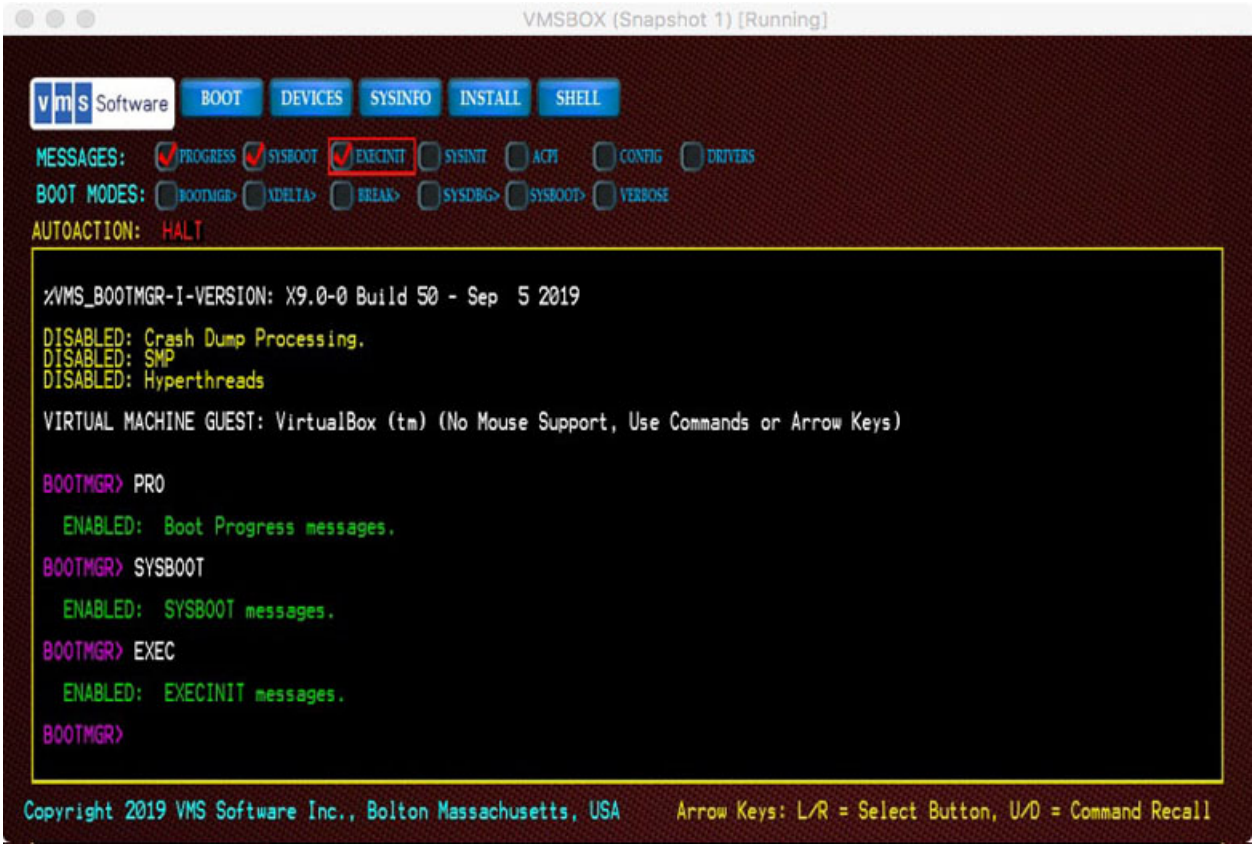
The definition of boot flags has changed for OpenVMS x86. Many of the earlier flags no longer applied to the x86 architecture or served functions that were unrelated to booting.

For OpenVMS on x86, we have reorganized the flags into three categories:

1. Message Controls

2. Boot Mode Controls

3. Debug and Diagnostic Controls



Typical Boot Message Control Flags

The Boot Manager presents graphical check-buttons for the most commonly used flags and commands to set and clear individual flags by name. Note however, that most Virtual Machine Hosts do not support mouse movement under UEFI, so if you wish to use the graphical buttons, you will need to press the arrow keys to select a button, then press ENTER to toggle the state of the button. As an alternative, you can use commands to control the flag states.

Each Boot Flag has a corresponding command to set or clear the flag. All such Boot Manager commands can be prefixed with **NO** to clear the flag. For example: **PROGRESS** sets the Progress Message flag and **NOPROGRESS** clears the flag. Most commands can be abbreviated to a minimum number of unique characters.

A **FLAGS** command is also available for managing the full set of available flags. Entering the FLAGS command without an additional value will display and interpret the current Boot Flag settings. In graphical modes, a chart of flag bits will also be displayed. To set or clear Boot Flags the desired HEXIDECIMAL value can be provided following the FLAGS command. For example, **FLAGS 1070** would set Boot Flag bits 4, 5, 6 and 12, enabling general boot PROGRESS messages and additional messages from SYSBOOT, EXEC_INIT and ACPI facilities.

When setting new flag values, be aware that the value you enter will supersede flags that had been set previously. For this reason, you may want to use the FLAGS command first to see the current value, then adjust the value to enable or disable the flag bit/s you are interested in.

The Boot Manager saves your Boot Flags value in response to a BOOT, EXIT, or SAVE command. Flag values that were set during a BOOT command will also be preserved as the default boot command string to be used on subsequent boots.

Chapter 16. Boot Manager Command Dictionary

The Boot Manager provides a set of commands that serve the needs of developers and system managers. The subset of these commands that support developers is subject to change and some of the commands will be disabled in production releases. This version of the Users Guide describes all of the commands that are active at the time of writing. Each command description will indicate its intended audience.

The command descriptions show the full length command verbs, followed by the abbreviated command verb shown in braces. In certain instances, commands that result in particularly significant operations may not support an abbreviated form.

COMMAND RECALL and LOGGING

The Boot Manager provides a limited command recall buffer that can be accessed with the UP or DOWN Arrow keys. The Boot Manager does not currently support scrolled output due to its synchronous operation and the undesirable overhead involved in providing unlimited scrolling. It is quite unusual for the early boot process to produce large amounts of text prior to transfer to OpenVMS, at which point the Operator terminal and logging are available. In cases where Boot Manager commands produce more lines of output than will fit on a single screen, the **PAGE** command will pause the output as text exceeds each visible display page.

In cases where a permanent log of Boot Manager output is desired, output can be directed to a COM Port and Terminal Emulator. These features are described in their respective command descriptions.

Push Buttons

When operating in Graphical mode, common commands are represented by rows of push buttons. If a pointing device (mouse or touch-pad) is active, the push buttons can be activated as you would expect for a graphical application.

If no pointing device is active, the push buttons can be selected using the <Right Arrow> or <Left Arrow> keys. The selected button will be outlined in red. To activate the selected button, press <ENTER> or <ESC> keys. If you press any other keys, input focus will return to the command line.

COMMAND SUMMARY

Boot Manager commands use a very simple syntax.

A command verb is followed by an optional parameter separated by a space.

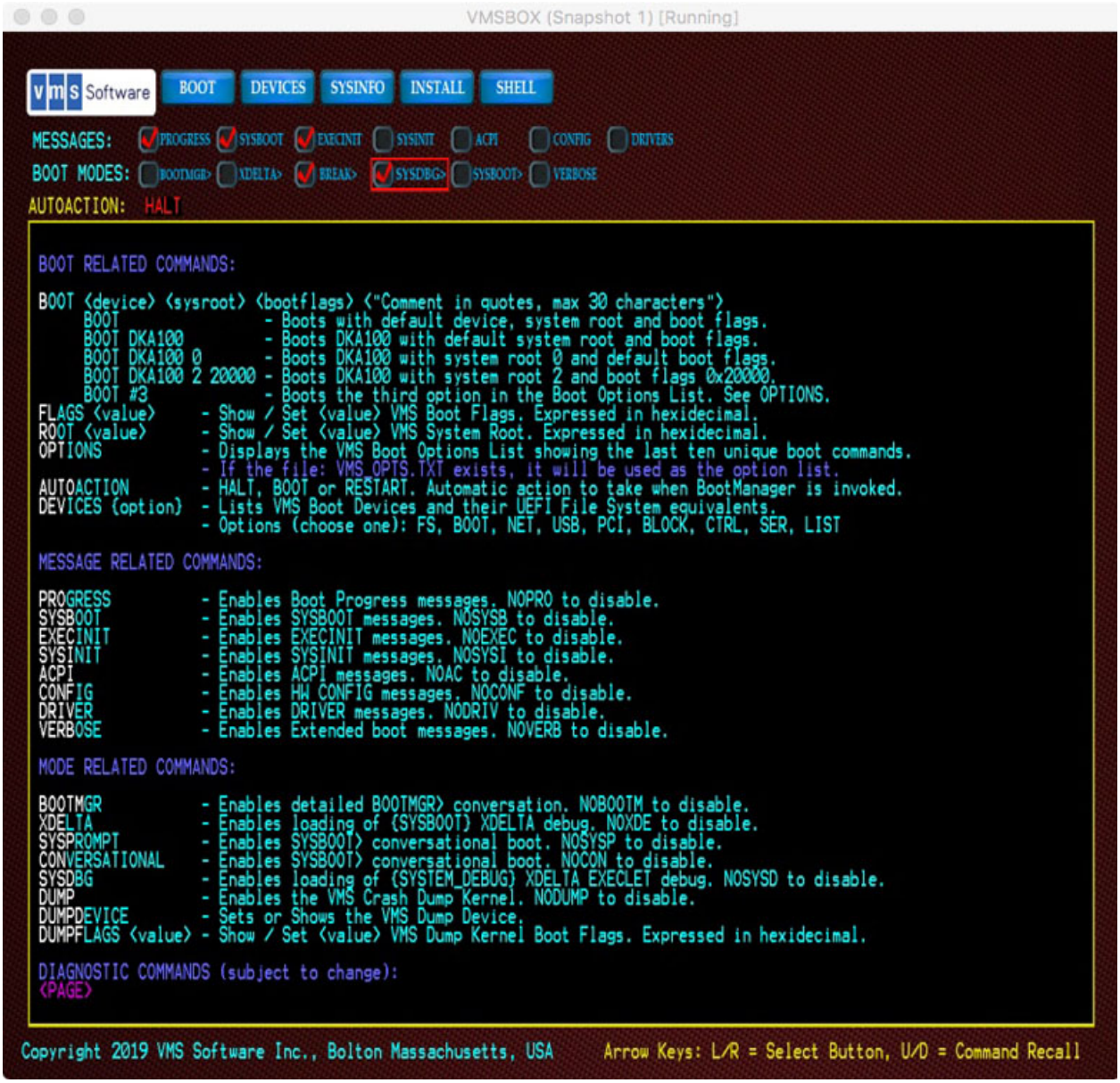
Command verbs can be abbreviated to the smallest number of characters required to assure uniqueness. Certain commands having significant effect, must be entered in full as a means of verifying user intent.

Commands that *SET* a flag or enable a feature, are typically *CLEARED* by prefixing the verb *NO* in front of the original command. For example, the command: *PROGRESS* enables a flag to produce boot progress messages. The command: *NOPROGRESS* clears the flag. Expressed another way; the Boot Manager does not use *SET, CLEAR, ENABLE or DISABLE* command precursors.

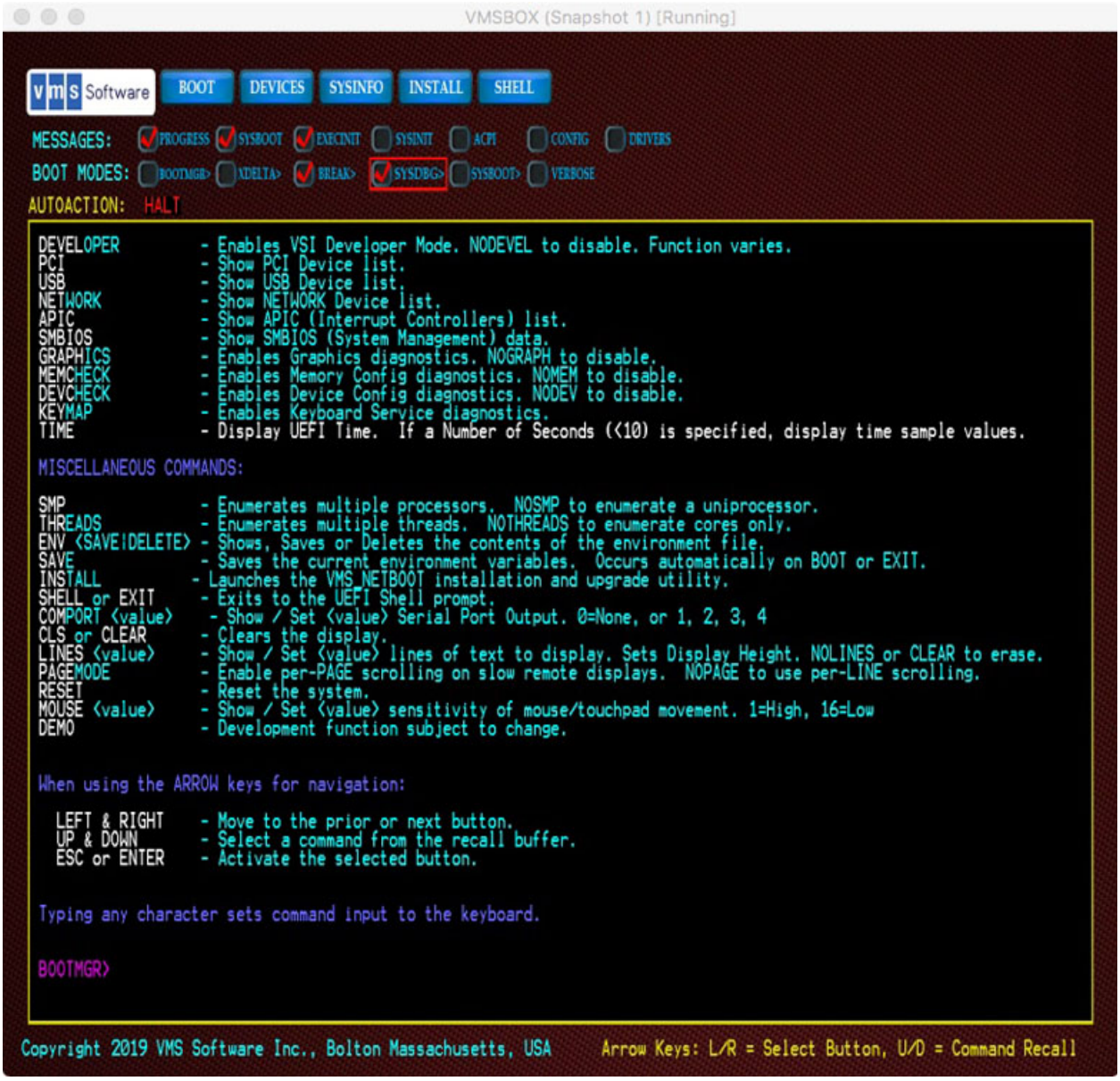
The following commands are available from the BOOTMGR> prompt.

HELP

HELP {H or ?}Displays a list of commands and describes their functions.



Boot Manager HELP (1 of 2)



Boot Manager HELP (2 of 2)

BOOT

BOOT {{device} {root} {flags}}

The BOOT command initiates a system boot. Several optional arguments are available.

If you have previously booted the system, a BOOT command with no additional arguments, will attempt to boot the system using the prior boot command string, including device, root and flags.

If you have not previously booted the system, a BOOT command with no additional arguments will attempt to boot the system from the device where the Boot Manager resides. In most cases, this will be your default system disk.

Optional Arguments

BOOT {device}

The first optional argument is an OpenVMS Device name that you wish to boot from. Unlike prior versions of OpenVMS, the Boot Manager can be run from one device and boot a different device. You

no longer need to figure out which UEFI file system device (fsx:) equates to your boot device. The Boot Manager will do this for you.

For example: BOOTMGR> **BOOT DKA100:**

Note

Mapping of OpenVMS to UEFI file system devices requires support of the UEFI Shell protocol. In some rare cases, this protocol is not available in platform firmware. In this case, you can use the UEFI file system device name (fs0: etc.)in the BOOT command.

You must choose one of the *bootable* devices listed by the DEVICE command. The listed devices are those which have passed a series of tests to determine that they contain bootable OpenVMS images. Specifying the trailing colon is optional. Note that the scope of devices that the Boot Manager sees during initialization is limited to those for which UEFI drivers are available. Complex storage systems do not always expose their devices to UEFI for booting. Most often, the listed devices will include the system boot device. Once OpenVMS has booted, it gains additional capability for detecting devices and in some rare cases, the OpenVMS device names may differ from the name that was used during boot.

BOOT {device} {root}

The second optional argument is a System Root to boot from. You can set the default System Root using the ROOT command, or you can specify it as the second argument in a fully qualified boot command.

For example: BOOTMGR> **B DKA0 0 1** Boots from DKA0 with the PROGRESS flag set and System Root 0.

BOOT {device} {root} {flags}

The third optional argument is a *hexadecimal value* representing *Boot Flags*. Refer to the description of FLAGS for more details. Boot flags may also be passed into the Boot Manager when it is launched from the UEFI Shell prompt. If you specify flags at the UEFI Shell prompt, prefix the flags with "-fl" as you would on earlier versions of OpenVMS.

For example:

Shell> **vms_bootmgr DKA0 –fl 0,807**

The above command boots device DKA0 using system root 0 and boot flags 0x807

You do not need the -fl prefix or comma when entering a BOOT command at the BOOTMGR> prompt. The equivalent command line would be BOOTMGR> **BOOT DKA0 0 807**

Note

The OpenVMS x86 Boot Flag definitions are different from prior versions of OpenVMS. Refer to the FLAGS command description for details.

BOOT OPTIONS LIST

As an alternative to entering boot command arguments, you can also boot from a pre-defined list of boot options. These are NOT the same as UEFI Boot Option variables. UEFI Boot Option variables get you to the Boot Manager, NOT to an OpenVMS boot device.

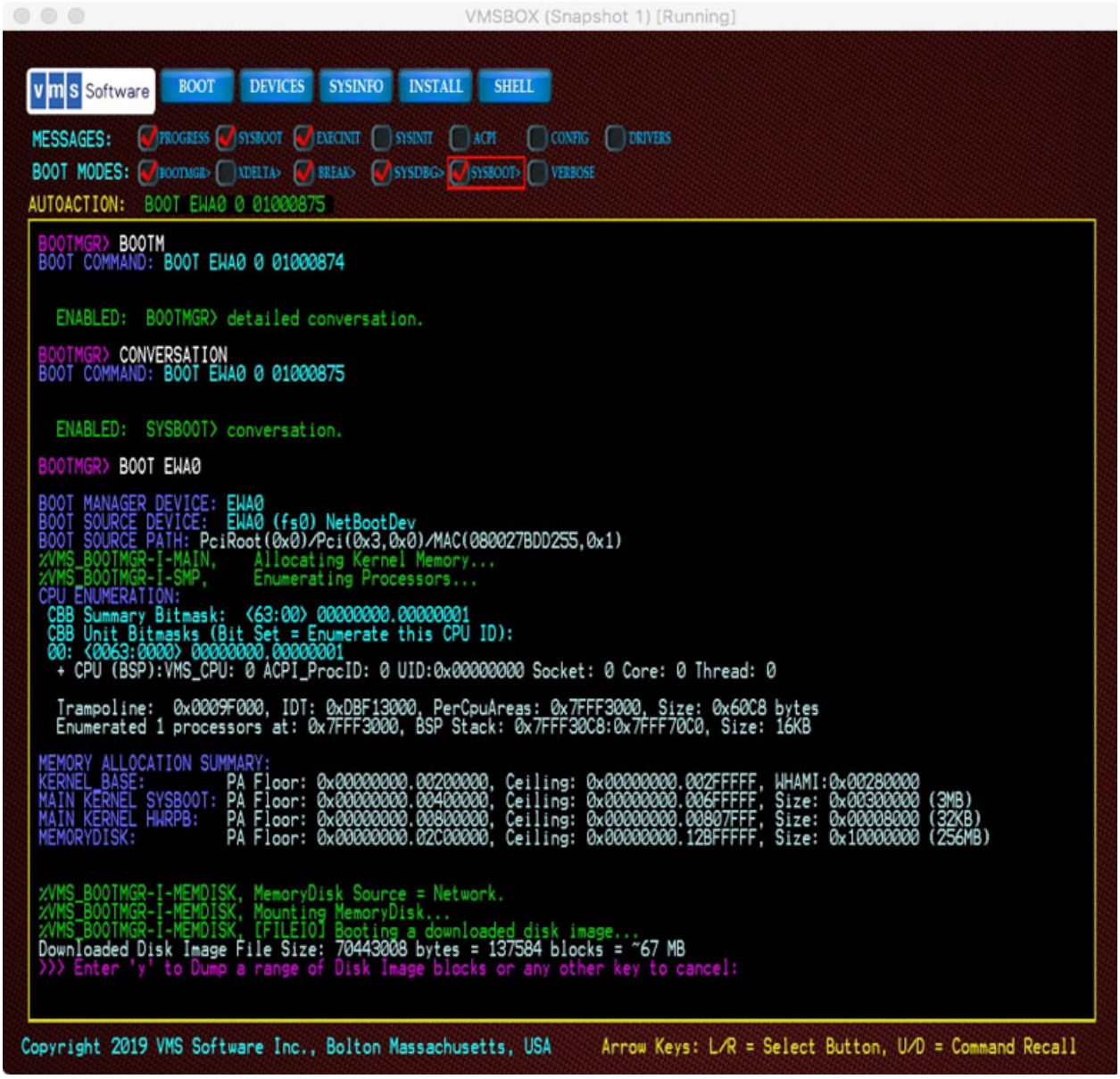
To use a predefined list of boot commands, you must first create the list. The list is a simple text file that can be created by the UEFI editor or copied to the system using FTP or other mechanism to get the file into the EFI partition.

Refer to the **OPTIONS** command for details on creating the Options List.

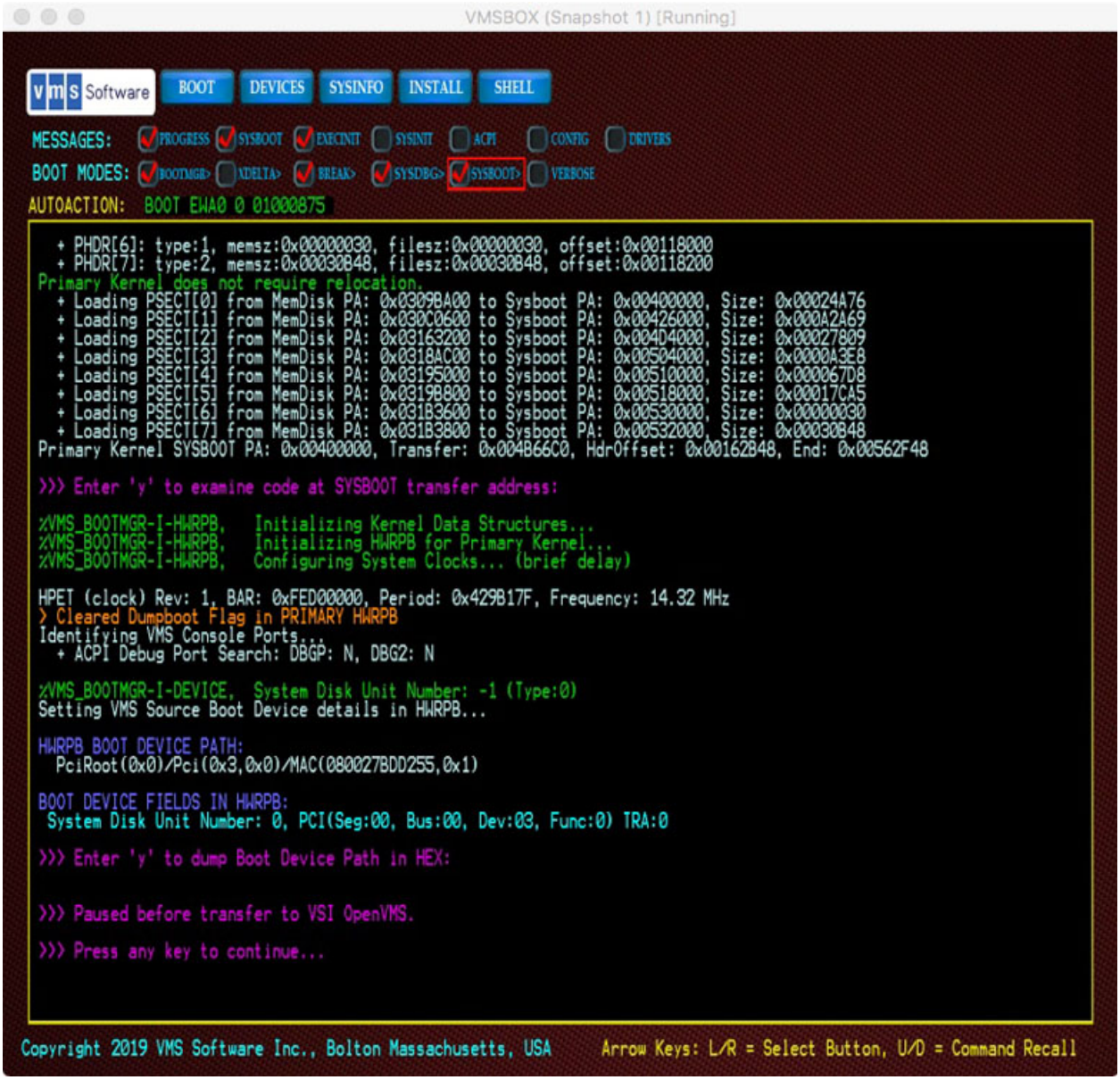
If the Boot Manager finds the options list, the OPTIONS command will enumerate and display the list.

To select a listed option, enter **BOOT #** followed by the option number in the boot options list.

This feature is intended to simplify the task of booting many system configurations for test purposes, but it may be useful for other reasons too.



Boot with Boot Manager Flag Set (1 of 2)



Boot with Boot Manager Flag Set (2 of 2)

AUTOACTION BOOT | HALT | seconds

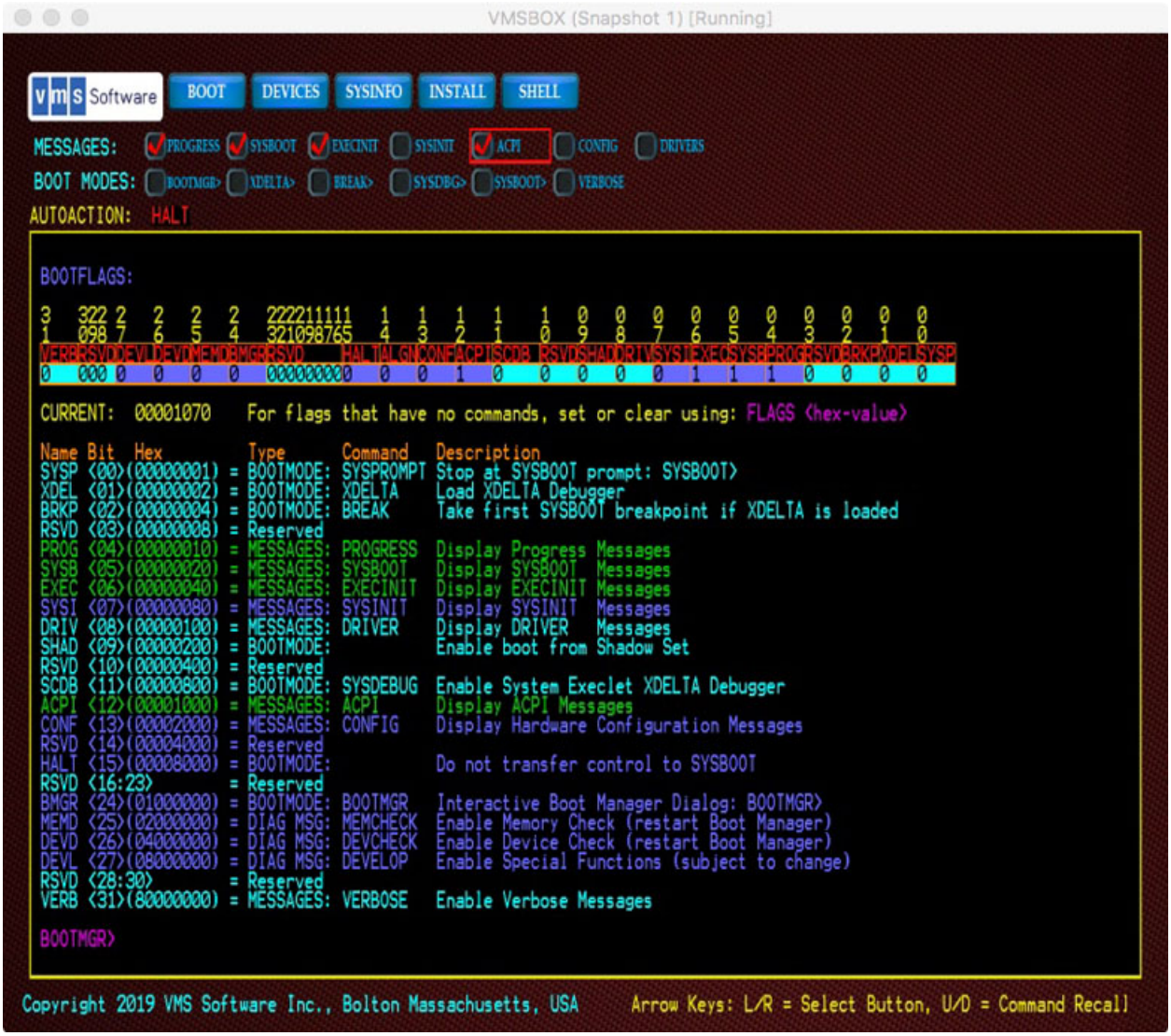
If you have previously set AUTOACTION to BOOT, the Boot Manager will provide a countdown prior to launching the default boot command (prior successful boot command). Pressing the Escape key during this countdown period will abort the countdown and present the BOOTMGR> command prompt. Pressing any other key during this countdown, skips the remaining count and commences with booting.

Refer to the AUTOACTION command description for more details.

FLAGS and MESSAGE CHECK-BUTTONS

FLAGS {<arg>}

The FLAGS command displays the current set of Boot Flags. Boot Flags are manipulated through commands and if in Graphical mode, buttons. The optional argument is a set of Boot Flags specified as hexadecimal value up to 32-bits wide.



The FLAGS Command showing all Boot Flag definitions.

Boot Flags can also be specified on the UEFI command line when the Boot Manager is launched.

For example: Shell> **vms_bootmgr DKA0 –fl 2,1** Sets the Progress Boot Flag (bit<0>) and sets System Root to 2.

Note that the X86 Boot Manager and OpenVMS V9.x for x86 use a different set of Boot Flags than prior versions of OpenVMS. In order to have better control over messages, each major phase of the boot process has been assigned its own flag. All phases are also affected by the VERBOSE bit<31> flag which greatly extends the type of troubleshooting messages that are displayed during boot and runtime.

Check Buttons

When operating in Graphical mode, common features are represented by rows of check buttons. If a pointing device (mouse or touch-pad) is active, the check buttons can be activated as you would expect for a graphical application.

If no pointing device is active, the check buttons can be selected using the <Right Arrow> or <Left Arrow> keys. The selected button will be outlined in red. To toggle the selected button, press <ENTER> or <ESC> keys. If you press any other keys, input focus will return to the command line.

Check buttons differ from push buttons because they visually represent two static states (set or clear). The state of the check buttons is preserved across boots. Each check button corresponds to a pair of commands to set and clear the represented state. These commands use the flag name as the command and the prefix “NO” to clear the flag.

For example:

BOOTMGR> **PROGRESS** Sets the Progress check button and flag.

BOOTMGR> **NOPROGRESS** Clears the Progress check button and flag.

Using text commands will toggle the visual state of the check buttons, and using a pointer to toggle the button state will alter the associated flag. The FLAGS command will display the current state of the various flags. Only the most commonly used flags have corresponding check buttons.

TOP ROW OF CHECK BUTTONS

The top row of check buttons represent the common messaging flags using during OpenVMS boot.

PROGRESS

The PROGRESS flag, when set, causes the Boot Manager and OpenVMS to display a number of progress messages during boot. These progress messages are carefully selected and formatted to provide a customer-facing indication of major boot phase transitions. The Boot Manager displays these messages in green.

The PROGRESS flag is Boot Flag bit<4> (Hex value 00000010).

NOPROGRESS

Disables the PROGRESS flag and corresponding messages.

SYSBOOT

The SYSBOOT flag, when set, causes the SYSBOOT component to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during the early boot phase. The Boot Manager displays these messages in light blue. The SYSBOOT flag is Boot Flag bit<5> (Hex value 00000020).

SYSBOOT is the earliest boot phase where paging, memory management and system cells are initialized and early execlets are loaded from the Memory Disk.

NOSYSBOOT

Disables the SYSBOOT flag and corresponding messages.

EXECINIT

The EXECINIT flag, when set, causes the EXEC_INIT components to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during the Executive Initialization boot phase. The Boot Manager displays these messages in light blue. The EXECINIT flag is Boot Flag bit<6> (Hex value 00000040).

EXEC_INIT is the boot phase where page tables are constructed, the remaining execlets are loaded from the system disk (no longer the Memory Disk), interrupts are enabled subsystems are initialized, DCL becomes available and symmetric multiprocessing (SMP) begins.

NOEXECINIT

Disables the EXECINIT flag and corresponding messages.

SYSINIT

The SYSINIT flag, when set, causes the SYSINIT process to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during this phase. The Boot Manager displays these messages in light blue. The SYSINIT flag is Boot Flag bit<7> (Hex value 00000080).

The SYSINIT process is the final boot phase where system and user processes are initiated and the swapper runs.

NOSYSINIT

Disables the SYSNIT flag and corresponding messages.

ACPI

The ACPI flag, when set, causes the ACPI components to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during this phase. The Boot Manager displays these messages in light blue. The ACPI flag is Boot Flag bit<12> (Hex value 00001000).

ACPI (Advanced Configuration and Power Interface) is the boot phase where system devices and their interrupt levels are initialized. Beware that the use of both ACPI and VERBOSE flags creates a huge amount of output.

NOACPI

Disables the ACPI flag and corresponding messages.

CONFIG

The CONFIG flag, when set, causes the IO Database components to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during this phase. The Boot Manager displays these messages in light blue. The CONFIG flag is Boot Flag bit<13> (Hex value 00002000).

CONFIG is the boot phase where peripheral devices are enumerated and drivers are installed. Beware that the use of both CONFIG and VERBOSE flags creates a huge amount of output.

NOCONFIG

Disables the CONFIG flag and corresponding messages.

DRIVER

The DRIVER flag, when set, causes various device drivers to display a number of detailed messages during boot. These messages are carefully selected and formatted to assist with troubleshooting problems during this phase. The Boot Manager displays these messages in light blue. The DRIVER flag is Boot Flag bit<8> (Hex value 00000100).

DRIVER flag provides message from any participating drivers. Not all drivers have implemented this flag.

NODRIVER

Disables the DRIVER flag and corresponding messages.

HALT

The **HALT** command causes the Boot Manager to halt just prior to transferring control to SYSBOOT. This allows the developer to work with Boot Manager behavior and to view various structures just prior to booting.

DUMPFLAGS

DUMPFLAGS {<arg>}

Similar to Boot Flags, this set of flags is passed to the Dump Kernel when it boots. This is to allow developers some further control over Dump Kernel boot behavior. Not all flags apply to Dump Kernel operation.

ROOT

ROOT {<arg>}

The ROOT command without any argument, displays the current default System Root being booted. The optional argument becomes the default System Root. This is a persistent variable.

OPTIONS

OPTIONS {<arg>}

As an alternative to entering boot command arguments, you can also boot from a pre-defined list of boot options. These are NOT the same as UEFI Boot Option variables. UEFI Boot Option variables get you to the Boot Manager, NOT to an OpenVMS boot device.

To use a predefined list of boot commands, you must first create the list. The list is a simple text file that can be created by the UEFI editor or copied to the system using FTP or other mechanism to get the file into the EFI partition.

The location and name of the file must be: **fsx:\EFI\BOOT\VMS_OPTS.DAT** where “fsx:” is the file system device where the Boot Manager resides.

The file can contain any number of boot command lines. Each command can be followed by a descriptive comment, prefixed by a semi-colon ;

For example:

BOOT DKA100: 0 1 ; Internal system disk, progress flag set, root 0

BOOT DKA100: 1 3 ; Internal system disk, progress and sysboot flags set, root 1

BOOT DGA0: ; Boot DGA0 with default flags and root

etc.

If the Boot Manager finds this file, the OPTIONS command will enumerate and display the list.

To select a listed option, instead of a boot device, enter “#” followed by the option number in the boot command.

BOOTMGR> **BOOT #3** will boot DGA0, the third entry in the list.

This feature is intended to simplify the task of booting many system configurations for test purposes, but it may be useful for other reasons too.

AUTOACTION

AUTOACTION {<arg>}

The AUTOACTION command determine what automatic action occurs when the Boot Manager is launched.

Arguments include:

HALT - Remain at the BOOTMGR> command prompt.
BOOT - Countdown before issuing the previous boot command.
seconds - Sets a countdown delay between 1 and 30 seconds.

If the boot countdown is interrupted by pressing the ESC key, the countdown stops and the BOOTMGR> prompt is displayed. Pressing any other key during the countdown skips the remaining countdown and proceeds to boot.

DEVICES

DEVICES {ALL | BOOT | FS | NET | USB | PCI | BLOCK | CPU}

The DEVICE command presents a list of qualified OpenVMS boot devices. The bootable devices are enumerated with OpenVMS device names.

The listed devices are those which have passed a series of tests to determine that they contain bootable OpenVMS images.

The tests include:

1. Does the device contain a GPT (GUIDed Partition Table)?

2. Does the device contain a properly configured set of three OpenVMS partitions?

3. Is there a UEFI partition among this set?

4. Does the UEFI partition contain \EFI\BOOT\VMS_BOOTMGR.EFI?

5. Do the partitions satisfy security checks?

Note that the scope of devices that the Boot Manager sees during initialization is limited to those for which UEFI drivers are available. Complex storage systems do not always expose their devices to UEFI for booting. Most often, the listed devices will include the system boot device. Note however, that as OpenVMS boots, it gains additional capability for detecting and enumerating devices and in some rare cases the OpenVMS device names may differ from the name that was used during boot.

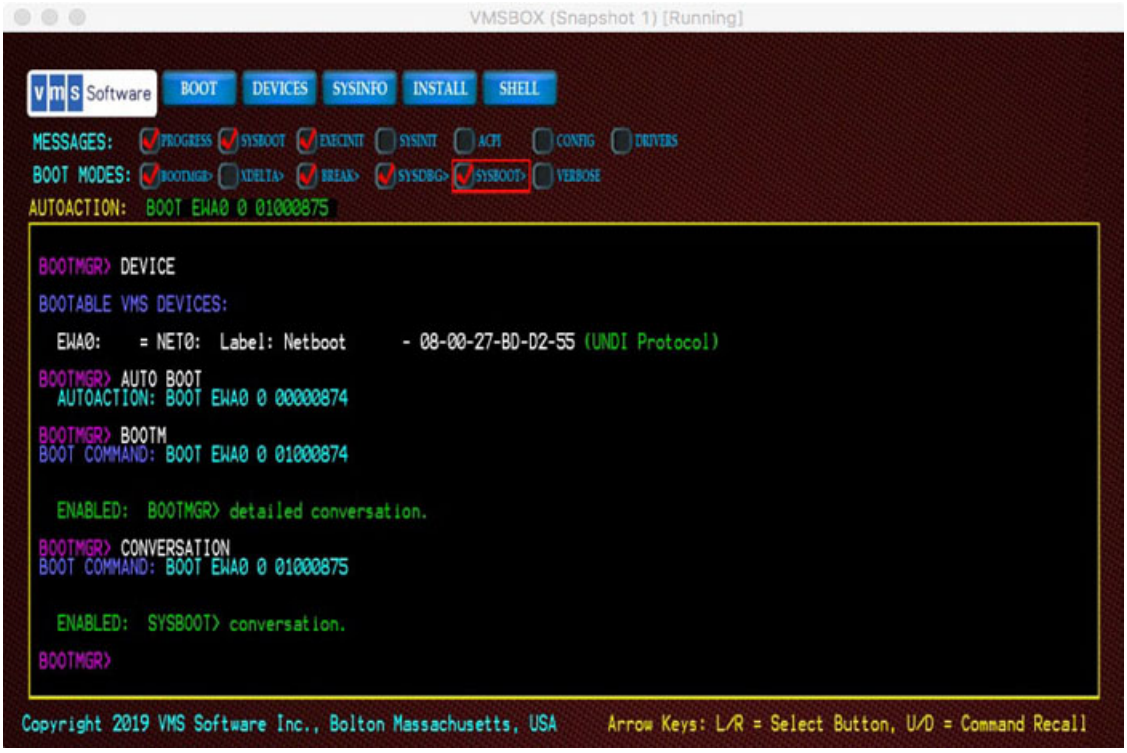
An optional argument can be provided to the DEVICE command to filter the type of devices shown.

Note

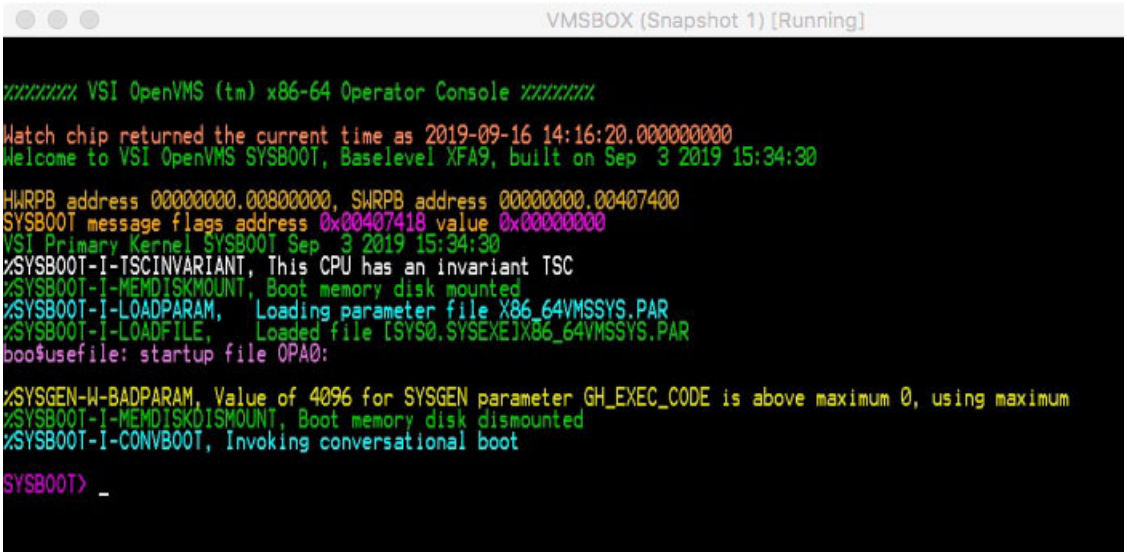
The **DEVICE CPU** option executes a Hardware Compatibility Test to verify that the system has the necessary features to run OpenVMS. It then continues to decode the CPU ID instruction to list the full feature set of the processor chip.

The following list of arguments can be specified:

ALL - Shows ALL device types whether bootable or not.
FS - Shows File System devices which have corresponding fsx: names, whether bootable or not.
BOOT - Shows devices which have been deemed bootable and have OpenVMS Device Names assigned.
NETWORK - Shows network devices. Those that support UNDI Protocol are considered bootable.
BLOCK - Shows devices that support the UEFI BlockIO Protocol.
USB - Shows USB devices. See also the USB command.
PCI - Shows PCI devices. See also the PCI command.
CPU - Decodes the CPU ID instruction and runs the Hardware Compatibility Test.



DEVICE, AUTO, and CONVERSATION Commands



CONVERSATIONAL BOOT

DEVELOPER (development feature)

The DEVELOPER command is reserved for Boot Manager development use. Its function is subject to change.

PCI

PCI

The PCI command displays the result of PCI and PCIe Bus Scans. Details of each device discovered is displayed. Where possible, vendor and device ID are decoded and a one-line description is displayed.

The vendor and device ID's are used for searching an internal database of known vendors and devices. If a match is found, the database information is displayed. If the vendor or device is unrecognized, the Boot Manager will list it as a "Curious Device" and any embedded identification strings in the device's configuration structures will be displayed.

Identification or failure to identify any given device does NOT imply that an OpenVMS device driver exists for the device or that the device is unsupported.

The Boot Manager's database can be updated periodically to pick up the most recent definitions from a PCI Device Registration website. This is a semi-automated procedure that requires engineering support. Therefore, it is expected to be updated just prior to each shipping version of OpenVMS.

USB

USB The USB command displays the result of a USB device scan. Details of each device discovered is displayed. Where possible, vendor and device ID are decoded and a one-line description is displayed.

The vendor and device ID's are used for searching an internal database of known vendors and devices. If a match is found, the database information is displayed. If the vendor or device is unrecognized, the Boot Manager will list it as a "Curious Device" and any embedded identification strings in the device's configuration structures will be displayed.

The Boot Manager's database can be updated periodically to pick up the most recent definitions from the USB Device Registration website. This is a semi-automated procedure that requires engineering support. Therefore, it is expected to be updated just prior to any shipping version of OpenVMS.

NETWORK

NET

The NET command displays information about each network device on the system. Bootable network devices are identified. The MAC address of each device is provided for configuring boot servers.

APIC (development feature)

APIC

The APIC command displays configuration details for each APIC (Advanced Programmable Interrupt Controller) in the system. Various types of APICs exist including IO-APICs, Processor-Local-APICs, X2APICS (large scale systems) and others. This information is useful for kernel and driver developers.

SMBIOS

SMBIOS

The SMBIOS command displays and interprets the various System Management BIOS tables. Interpretation is provided according to the official SMBIOS specification. Many vendor-specific tables exist. These are displayed in a raw data form.

GRAPHICS (development feature)

GRAPHICS

The GRAPHICS command enables diagnostics functions in the Graphical mode of operation. Details about the graphical framebuffer and display geometry are provided, then the current pointer coordinates and button-press events are displayed assuming their movement is reported by UEFI firmware.

NOGRAPHICS

Disables the Graphical mode diagnostic messages.

MEMCHECK (diagnostic feature)

MEMCHECK

The MEMCHECK command enables extended diagnostics regarding memory allocation and assignment of memory to OpenVMS during boot. Note that this command consumes additional memory and in some cases, may prevent successful boot.

NOMEMCHECK

Disables the MEMCHECK diagnostics.

DEVCHECK (diagnostic feature)

DEVCHECK

The DEVCHECK command executes a PCI/e Bus Scan and enables extended diagnostics regarding device discovery and assignment of OpenVMS device names.

This is a one-shot command which does not require a NODEVCHECK command.

KEYMAP (development feature)

KEYMAP

The KEYMAP command provides the early developer with interpretation of Key Codes and Unicode characters as they are entered. The BOOTMGR boot flag must also be set to use this diagnostic function.

TIME

The TIME command with no argument will display the current UEFI Time.

Specifying a number of seconds between 1 to 10 after the TIME command will cause the Boot Manager to issue a delay of that many seconds to each of the viable system clock candidates, displaying their respective counter values. To set the UEFI time, use the Shell> **time hh:mm:ss** command.

SMP {num-APs} (development feature)

The SMP command enables multiple processor core enumeration.

SMP support is available in V9.0-D, but is disabled by default. To enable SMP, you must first select additional processors in your Virtual Machine Guest setup, then issue the SMP command in the Boot Manager prior to booting. Once you have issued the SMP command, it should remain enabled until you issue a NOSMP command. During boot, you should see a startup message from each secondary processor (also know as Auxiliary Processors or AP's).

Specifying a number after SMP will limit enumeration to that number of Application Processors (AP's). If no number is specified, the full set of AP's will be enumerated.

NOSMP

Limits the system to a single processor.

THREADS (development feature)

The THREADS command enables processor hyper-thread enumeration. **For the V9.0-C EAK threads are not enumerated. OpenVMS does not by default use hyper-thread features due to certain architectural security issues, but in an environment where the security concerns are well understood, these additional Application Processors can be used.**

NOTHREADS

Disables thread enumeration.

ENV

ENV {SAVE | DELETE}

The ENV command displays a list of currently defined OpenVMS environment variables. If the optional argument is SAVE, the current set of values is saved to Non-Volatile storage and will be restored the next time the Boot Manager is run. If the optional argument is DELETE, the environment variables will be deleted and their variables will take default values.

Environment variables which have changed are automatically saved whenever a BOOT, EXIT, or SHELL command is issued.

Important Note: UEFI Environment Variable storage is used whenever possible. However, some Virtual Machines fail to provide persistence of variables across power cycles. The Boot Manager attempts to store its set of environment variables in both the UEFI Non-Volatile storage and a file in the UEFI partition named: VMS_ENV.DAT. This file is in binary form with security features. If the user attempts to edit the file, it will be deleted by the Boot Manager and a new file will be created. Note too that this creates a requirement for the boot device to be writable. During installations, typically from a DVD or over a network, the environment variables will not be stored.

SAVE

SAVE

Forces the save of current environment variable values.

INSTALL

INSTALL

The INSTALL command and its associated Push-Button is intended to initiate various installation and update procedures.

This feature has not yet been fully implemented.

SHELL and EXIT

SHELL or EXIT

The SHELL and EXIT commands exit the Boot Manager and return to the UEFI Shell prompt.

COMPORT

COMPORT {<arg>}

The COMPORT command used with no argument will display the currently active serial COM Port Numbers (0 through 4).

Specifying **COM 0** will disable all serial port output. Specifying a value of 1 through 4 will enable COM 1 through COM 4 output respectively. More than a single port can be enabled. These are the x86 architecturally defined serial ports.

If the Boot Manager detects a terminal connection, it will assign it to COM 1 (port address 0x3F8). Specifying a COM port causes the Boot Manager output to be directed to that serial port. If no COM command is issued, OpenVMS will still support terminal connections. Only the Boot Manager output will be excluded from the serial port device.

CLS or CLEAR

CLS & CLEAR

Each of these commands clears the current Graphical or Rendered-Text mode display.

LINES

LINES {<arg>}

When running the Boot Manager from within another system's window, such as when using a Virtual Machine Host window to launch OpenVMS as a Guest OS, the Boot Manager may not recognize the size of the parent window. In this case, several scroll region lines may fall beyond the visible window, requiring the user to scroll the parent window. This is a bother…

To solve this, the LINES command will display line numbers along the left of the visible scroll region. Simply look at the highest visible line number and use it as the optional argument for the same command.

For example: BOOTMGR> **LINES 30** Will resize the Boot Manager to display 30 lines. This is a persistent variable. To return to the maximum number of lines, simply enter an overly-large number of lines and the Boot Manager will adjust to the maximum lines that can be displayed in the current monitor.

For example: BOOTMGR> **LINES 100** Will typically result in a line count of about 45 lines.

NOLINES

Removes the line numbers displayed.

Note

The Boot Manager itself will only allow specifying the vertical size by number of lines. It does not support horizontal scaling. VirtualBox provides a menu item which allows the Boot Manager display to be freely scaled in it's parent window in both horizontal and vertical dimensions. To make the display resizeable under VirtualBox, look in the very top VirtualBox menu bar and select: *View->Scaled Mode*. You can now scale the BootMgr dialog as needed by dragging the dialog from its frame corner

PAGEMODE

PAGEMODE

When accessing the Boot Manager over certain networked interfaces, such as web-browser based management applications, text output may scroll unbearably slow. The PAGEMODE command causes

the Boot Manager to clear the screen after each page of output is displayed so as to avoid scrolling. This dramatically improves remote display performance.

As each page is displayed, the tag <PAGE> will occur at the bottom of the scroll region. Press the <ENTER> key to display the next page.

The Boot Manager will automatically use PAGE mode for command output that spans multiple screens, such as the HELP command.

NOPAGEMODE

Turns off the PAGEMODE display method and returns to normal line scrolling.

RESET

RESET

The RESET command issues a system reset. This differs from an EXIT command in that the platform firmware will reinitialize the system and clear memory.

DEMO

DEMO

The DEMO command takes on various functions as needed. Typically this is used to demonstrate some feature of the Boot Manager for other developers to evaluate.

Chapter 17. Device Enumeration

Scope of Device Scan

A design goal of the Boot Manager is to support OpenVMS Device Names when booting. This avoids the need to associate a UEFI File System device (i.e. FS0:, FS5: etc.) with an OpenVMS device. You can still specify a UEFI File System device at the BOOT command if you choose to (i.e. BOOTMGR> BOOT FS0) but you can also use the enumerated OpenVMS device name (i.e. BOOTMGR> BOOT DKA100)

It is important to note that UEFI firmware has limited visibility into complex storage configurations. Your boot device must fall within the scope of devices that are visible to UEFI firmware. It is entirely possible that devices which reside on complex storage arrays may not be visible to UEFI and only be discovered once OpenVMS has found them and loaded drivers for them.

In the prior Itanium architecture, you needed to select the UEFI File System device before invoking the vms_loader.efi utility. This is not the case for the x86 architecture. The OpenVMS Boot Manager is a completely stand-alone utility which is not tied to a specific version of OpenVMS. You can run any version of the Boot Manager from any UEFI file system device and invoke a boot from a different device. The only reason to explicitly specify a UEFI File System device first is if you want to use a specific Boot Manager or startup procedure located on that device. While Boot Manager versions may be compatible as far as basic booting goes, they will evolve to provide additional features.

When the Boot Manager evaluates devices, it looks for those devices which support UEFI FileIO protocol. It then probes each of these devices to see if the device contains an OpenVMS UEFI Partition and that the partition contains the necessary files to boot OpenVMS and it passes a few other integrity checks. Only after passing these validation steps, does the Boot Manager attempt to enumerate the device with an OpenVMS device name. Devices that fail validation will still be displayed as BlockIO devices, but are not considered bootable.

The command BOOTMGR> **DEVICE** will list bootable devices. Add one of the following qualifiers to filter the list of devices.

FS - Lists all disk devices supporting FileIO Protocol (whether or not bootable)
BLOCK - Lists all disk devices supporting BlockIO Protocol
BOOT - Lists all devices which contain bootable OpenVMS images
NETWORK - Lists all network devices (whether or not bootable)
USB - Lists all USB devices
PCI - Lists all PCI devices
LIST - Display extended info about all devices
CPU - Display extended info about the processor chip and perform a Hardware Compatibility Check.

The separate commands: USB and PCI provide additional details of these bus types.

When network devices are displayed, those that indicate support of the UNDI Protocol (Intel's Universal Network Device Interface) are also considered bootable.

The **DEVCHECK** diagnostic command will re-execute a PCI/e Bus Scan and display details of device enumeration.

OpenVMS Device Naming Peculiarities

OpenVMS Device Naming is somewhat complicated. Over the years, many special rules have crept into the enumeration scheme. It is a goal to have the Boot Manager enumerate devices in accordance with the way the operating system does, but there are some unavoidable exceptions to this which may result in a boot device changing names once OpenVMS has re-enumerated devices. For example, DKA0 may become DKA100 once OpenVMS applies certain topology rules. We are doing our best

to coordinate these procedures, but it will take extensive testing once we are at a point where more bus types are supported.

You can also specify a boot device using its UEFI Shell designation (fs0:, fs1: etc.). Use this method if there are problems with OpenVMS device naming in the Boot Manager.

Virtual Machines have introduced yet further difficulty in device naming. As one example, a USB-based DVD Reader may appear as a SATA disk under Virtual Box. Therefore, instead of assigning an OpenVMS USB device name which always begins with *DN* it will receive a name prefixed with *DK*. There are certain to be additional peculiarities as we move forward.

Network devices are particularly difficult to enumerate because the various device name prefixes are not based on features which are accessible through firmware methods. The current Boot Manager implementation enumerates all network devices as UEFI devices *NET0, NET1, etc.* These devices will take on other names once OpenVMS is booted.

USB Device Enumeration

USB devices have their own set of issues when it comes to assigning OpenVMS device names. When USB was first supported by OpenVMS, no effort was made to map the device name to the physical topology of the device. Instead, a simple sequential number was assigned to each device and the UCM (USB Configuration Manager) utility was created to match devices, drivers and unit numbers based on embedded serial numbers of the devices. This was perhaps not the best way to go and it failed to consider issues like devices which failed to implement embedded serial numbers.

Assigning USB device names based on physical topology solves some of the naming issues, but introduces a few new ones. For example, the USB architecture supports up to seven levels of hubs, but the spec goes on to state that you would most certainly run out of power long before reaching such a full configuration. It is unlikely that OpenVMS customers would configure more than a small hierarchy of USB hubs and devices. Supporting a topology-based naming scheme capable of representing the full hierarchy would require significant changes to key data structures, but we can accommodate up to four levels of hub hierarchy within the bounds of existing structures.

The Boot Manager has implemented this type of topology-based USB device naming. The implementation under OpenVMS will evolve to match in a forthcoming production release. USB is not officially support on the V9.0 EAK.

Typical USB device names may appear as follows:

DNA0 - Storage device on USB controller A, Hub L0 (implied), End Port 0
DNB7 - Storage device on USB controller B, Hub L0 (implied), End Port 7
DNA14 - Storage device on USB controller A, Hub L1 Port 1, End Port 4
DNC024 - Storage device on USB controller C, Hub L2 Port 0, Hub L1 Port 2, End Port 0
DNA7777 - Storage device on USB controller A, Hub L3 Port 7, Hub L2 Port 7, Hub L1 Port 7, End Port 7

The last example above, illustrates the maximum four-level hub hierarchy (per controller) that can be accommodated by this scheme and existing data structures. There are pros and cons of implementing a topology based scheme for USB, but this one is closer to the way other devices are enumerated and it assures that a device inserted into a specific port will always receive the same device name. As always, OpenVMS will assign logical names and prefixes to these names with additional details if the device is cluster mounted. VSI is evaluating the role of the UCM utility. This may evolve into a tool that is only used if you choose to cluster-mount a USB device.

Part V. Virtual Machines

Using OpenVMS as a Virtual Guest

Table of Contents

Chapter 18. Virtual Machine General Information

The following sections describe the basic requirements for installing OpenVMS as a Guest OS on one of the supported Virtual Machine Hosts. It would be impractical to describe all of the details of Virtual Machine Host operation under the various operating systems, so this guide assumes that you are familiar with your chosen Virtual Machine product. Refer to your Virtual Machine product documentation as necessary.

Virtual Disk Formats

Several disk formats are used by the various Virtual Machine Hosts. Not all of these formats are compatible among hosts, so it is important to decide which format best suits your environment.

This guide focuses primarily on Oracle's VirtualBox Host. VirtualBox has its own **VDI** format, but this is not transferrable to other VM Hosts without applying conversion tools. The **VMDK** is transferrable (or easily convertable) between VirtualBox and VMware Hosts. VMDK format provides some additional features that we can take advantage of, therefore, this is our most commonly used format. KVM favors the **QCOW2** format which is also less portable among hosts.

Within the various formats, virtual disks can be dynamically sized or fixed size. A fixed size disk offers slightly higher performance because it does not need to expand as space is required, but a fixed size disk will immediately consume the maximum number of bytes. A dynamically sized disk will expand as needed, up to a defined maximum size. This is a little slower and also prevents some of the more useful conversions that we have found to be handy.

A Dump Disk and User Disks may benefit from being fixed size if the intent is to move them between Virtual Machines and other physical systems. For example if you are developing on an OpenVMS Itanium system. Of course, using FTP to move files between systems is a more effecient solution as opposed to moving entire disks!

Fixed size VMDK disks are handy because the actual VMDK file is a simple ASCII wrapper around the raw disk image (.DSK) file. OpenVMS users can use the LD Utility to create the .DSK file, then INIT and MOUNT it to allow copying files to/from the logical disk. The .DSK file can then be converted to a VMDK. Once the .VMDK file exists, you can still copy the .DSK back and forth to other OpenVMS systems without having to do the VMDK conversion again. If the disk were instead, dynamically sized, this would not be possible.

All of the supported VM Hosts will recognize a .ISO formatted storage device with no need for conversion.

Virtual Disk Format Conversions

The various Virtual Machine Hosts provide utilities for converting Virtual Disk Formats. Refer to your VM product documentation or search the internet for specific details and examples.

Below are a few Virtual Disk conversions we use most often:

1. OpenVMS.DSK (RAW) to VirtualBox (VMDK) fixed size:

   On your VirtualBox Host:

   vboxmanage internalcommands createrawvmdk -filename outfile.vmdk -rawdisk infile.dsk

2. VirtualBox (VMDK) to fixed size OpenVMS.DSK (RAW):

   On your VirtualBox Host:

vboxmanage clonemedium disk infile.vmdk outfile.dsk --format raw --variant Fixed

3. VirtualBox (VMDK) to KVM (QCOW2):

   On your KVM Host:

   # qemu-img convert -O qcow2 infile.vmdk outfile.qcow2

General Guest Installation Overview

Note

VSI has chosen to distribute the V9.0-x EAK releases as **pre-configured Virtual Appliances**. Virtual Appliances provide the quickest and easiest way to achieve a ready-to-use system. When using the Virtual Appliance, your system will be pre-configured, so this section can be viewed as reference material.

If you are not using the Virtual Appliance, then you need to decide whether to install OpenVMS from a network and Web Server or from a Virtual or Physical DVD.

The installation requires that you establish a Virtual Machine Guest with the attributes listed below and further defined in the subsequent product-specific sections.

General Guest Attributes

1. A UEFI interface and Shell application.
2. A recommended minimum of 8GB Memory
3. A Network Adapter
4. An acceptable Clock Source (HPET is preferred)

UEFI Interface: OpenVMS installation and boot requires the execution of UEFI (aka console firmware) applications. Each of the supported Virtual Machine Hosts provide an optional UEFI interface and Shell application. The steps required to enable the UEFI interface are provided in the respective sections.

Memory: The Guest should provide a minimum of 8GB of memory. 6GB is the very lower limit. When installing OpenVMS over a network and Web Server, the entire installation kit disk is downloaded into memory, so providing ample memory during installation is desirable. After installation, memory can be reduced. Additionally, if you are using the Dump Kernel to process any crash dumps, it is best to provide at least 512MB of additional memory.

Network Adapter: Installation over a network and Web Server requires an active network connection. The installation utility (VMS_KITBOOT.EFI) will issue a DHCP request to configure an adapter for your network. You will need to know the IP Address of your Virtual Machine Host and your Web Server. If you have chosen to install from a DVD, you should still set up a network adapter for remote terminal access.

Clock Source: Virtual Machines are notorious for having poor clock accuracy. OpenVMS V9.0 is aware of being booted as a Guest OS and attempts to identify the best available clock source. The most accurate clock source is the High Performance Event Timer (HPET), but this clock is not available on all Virtual Machine Host products, therefore OpenVMS will fallback on alternative clock sources (typically a LAPIC clock) if HPET is not present or not enabled.

If you have chosen to boot from a DVD rather than a Web Server, you will need to configure a DVD drive for your Guest and you can skip the following details pertaining to VMS_KITBOOT. The DVD will directly boot into the OpenVMS Boot Manager.

If you have chosen to boot the installation kit from a Web Server, VMS_KITBOOT.EFI must be executed to initiate the network installation. This utility is provided on the VSI FTP site as a Virtual Disk suitable for your chosen Virtual Machine product. You can FTP the Virtual Disk file to your Guest directory/folder or transfer it to a USB stick, whichever works best for your environment. Once the Virtual Disk is available to your Guest, you should be able to invoke VMS_KITBOOT from the UEFI Shell> prompt.

Chapter 19. VirtualBox Guest

Creating a VirtualBox Guest

If you have not already installed Oracle VirtualBox, download and install the appropriate version for your host operating system from *www.virtualbox.org*

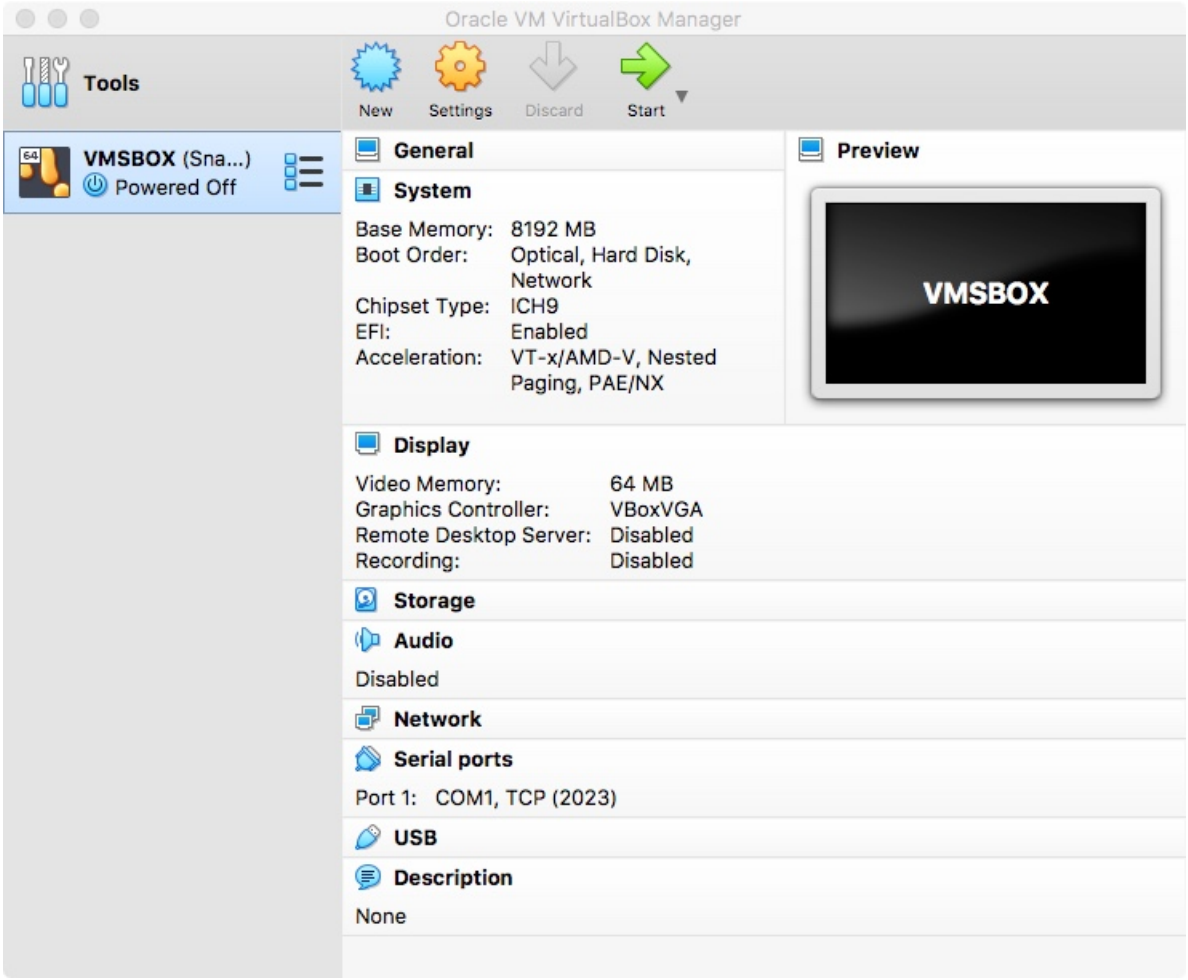
Optionally, you can download and install the **Guest Extension Pack.**

Note

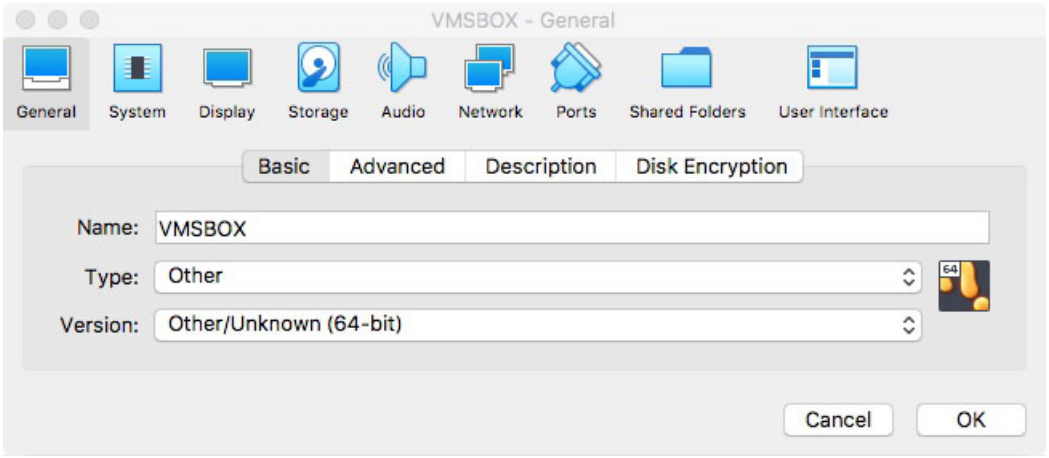
Although not strictly required, we have found it useful to install VirtualBox in an account with administrator privileges.

VirtualBox users should be aware that your mouse cursor will disappear when you click inside the Guest window. Unfortunately, the current version of VirtualBox does not report mouse movement when running in UEFI context. To regain your cursor, press the **Host** key. On a Windows based Host, this is the **Right CTRL** key, and on an Apple MAC OS-X Host it is the **Command** key. Use the Arrow and Enter keys to navagate the graphical Boot Manager interface.

Launch VirtualBox and create a new Virtual Machine Guest with the attributes listed in the following pages.

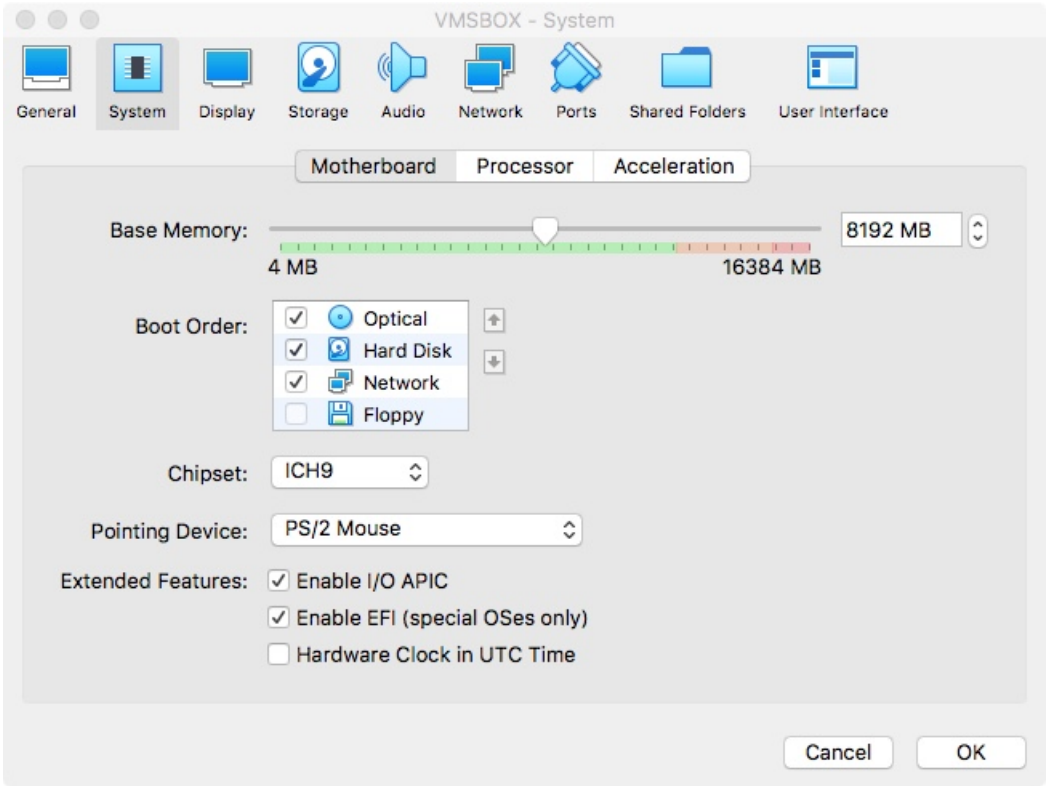


VirtualBox Guest Setup Screen



General Setup Screen

1. Name: Give your Guest a suitable name.
2. Type: OTHER
3. Version: Other/Unknown (64-bit)



System (Motherboard) Setup Screen

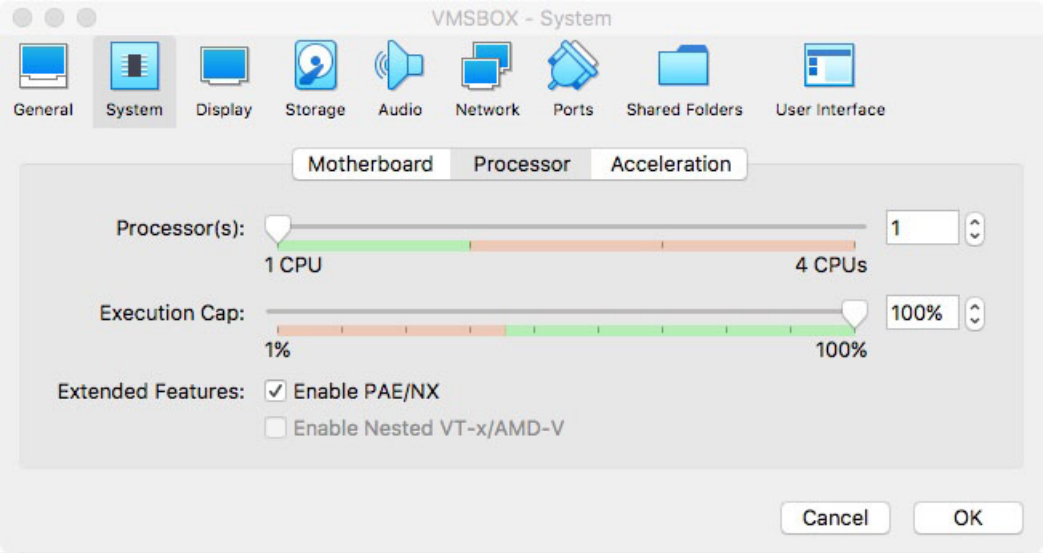
1. Base Memory: 8192MB (8GB recommended)
2. Boot Order: Optical, Hard Disk, Network
3. Chip set: ICH9
4. Pointing Device: PS/2 Mouse (see note below)

5. Extended Features: Enable I/O APIC, Enable EFI

Of particular interest is the *Enable EFI* attribute. This must be enabled to boot OpenVMS.

Note

VirtualBox does not currently support Mouse or Pointer movement for UEFI applications such as the OpenVMS Boot Manager.

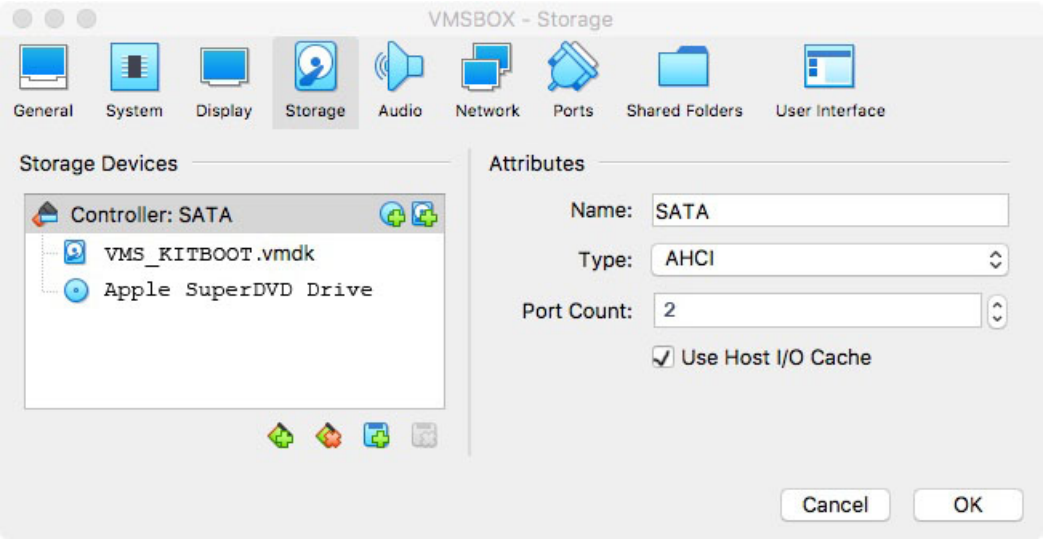


System (Processor) Setup Screen

1. Processors: 1 (OpenVMS V9.0-D supports SMP. To enable SMP, select additional processors and issue the **SMP** command in the Boot Manager prior to booting).

2. Execution Cap: 100%

3. Extended Features: Enable PAE/NX



Storage Setup Screen

1. Name: SATA (OpenVMS V9.0 EAK requires a SATA Controller)

2. Type: AHCI

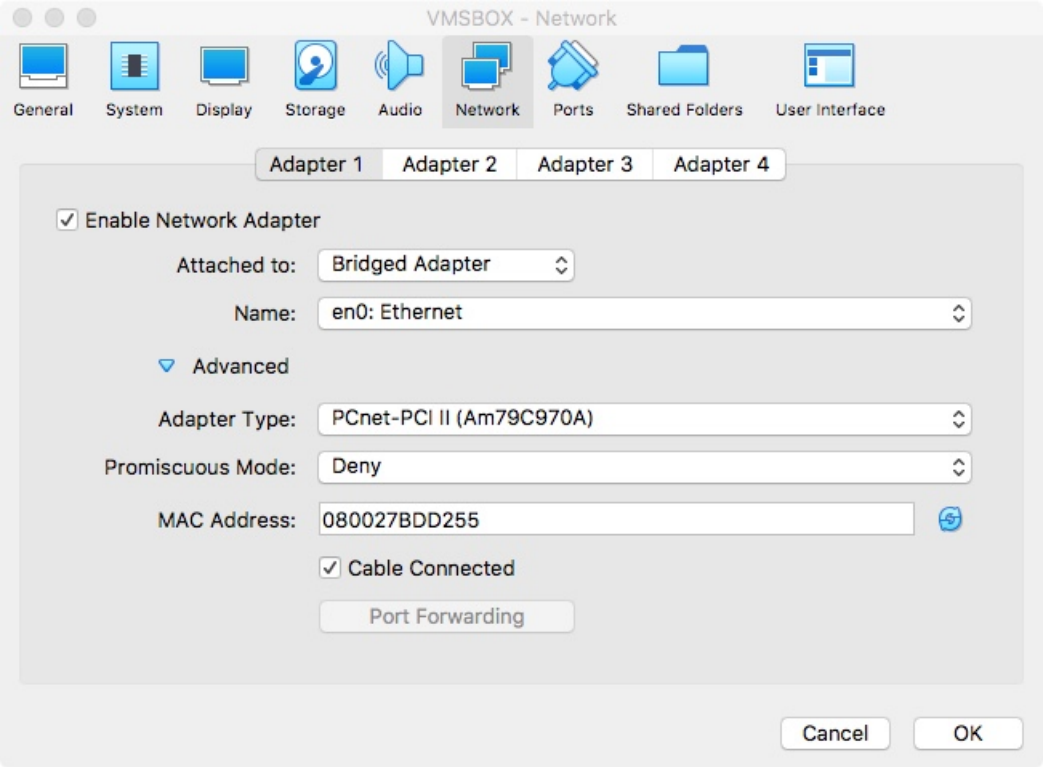
3. Use Host I/O Cache: Yes

If using Web Server booting, connect an existing Virtual Hard Disk to your SATA Controller selecting the Virtual Disk that contains the VMS_KITBOOT application (downloaded to your Host from the VSI FTP site). Because this virtual disk is a FAT formatted device, you can connect it as an IDE disk to avoid having OpenVMS see it as a SATA disk and name it DKA0.

Right click on the **SATA** Controller and select **Add Hard Disk** then **Choose Existing Disk**.
Add and select your virtual disk (vms_kitboot.vhdk). When you press **OPEN**, it should access the disk.

If you have chosen to install the OpenVMS EAK from a DVD, then insert the DVD in a reader connected to your host and select **Add Optical Disk** and select your DVD Drive.

If you have chosen to install the OpenVMS EAK from a Virtual DVD (file), then select **Add Optical Disk** and select your DVD file.



Network Setup Screen

1. Enable Network Adapter: Yes

2. Attached To: NAT (or Bridged Adapter. See Note)

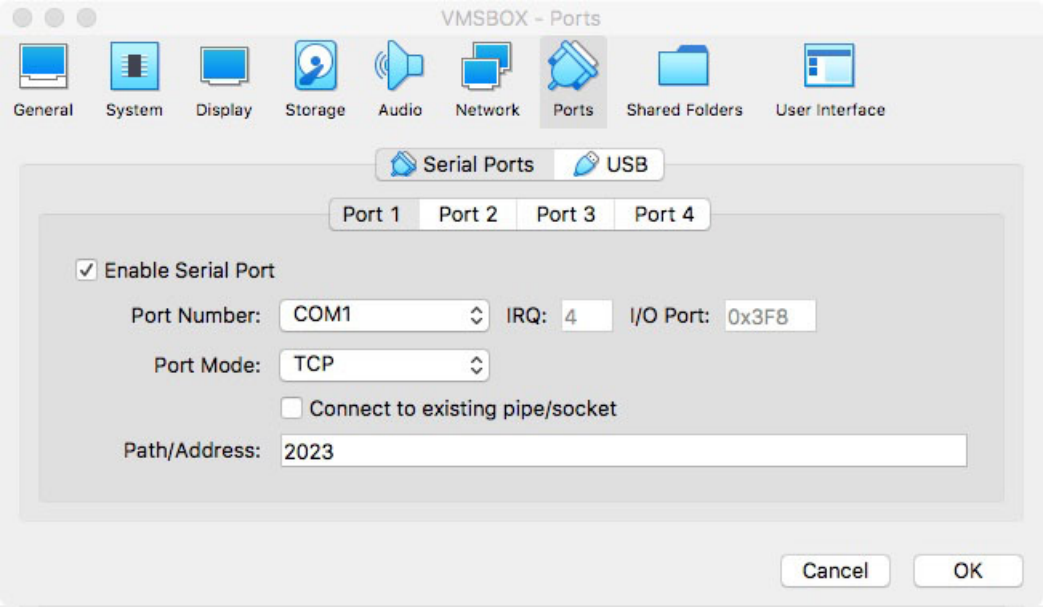
3. Name: en0: Ethernet

4. Adapter Type: Intel PRO/1000 MT Desktop

5. Promiscuous Mode: Deny

Note

The choice between Bridged Adapter versus NAT depends on your local network configuration and whether you are using a VPN. Consult your local network administrator or just try each to see which one works best for your network.



Serial Port Setup Screen

1. Enable Serial Port: Yes (Port 1)
2. Port Number: COM1
3. Port Mode: TCP (See note)
4. Connect to Existing Pipe/Socket: NO
5. Path/Address: 2023 (or your choice)

The COM 1 port 2023 will become OPA0: once booted. If you would like to have an additional TTA0: terminal session, define COM 2 using port 2024. Be sure to specify the correct port when connecting your terminal sessions.

Note

OpenVMS V9.0 EAK does not support a graphical subsystem (beyond the Boot Manager) so you are required to access the system through a serial port connection with a Terminal Emulator (telnet or SSH protocol). The setup suggested above directs all COM1 port (Operator and User) traffic to the network on port 2023. A Terminal Emulator (such as PuTTY) or Telnet/SSH session would connect to the Guest using the IP Address of the Host machine on TCP Port 2023. Be aware that the Virtual Appliance VSI distributes defines two serial port devices. If you switch to an alternative such as named pipes, be sure to switch both port settings.

Tips for setting up remote connection methods are provided in later sections of this document.

VirtualBox supports many more features than it exposes through its graphical user interface. From a command prompt on your host system enter the command: **vboxmanage help** to see a list of additional management commands.

One such command will enable the use of the High Performance Event Timer (HPET) as the primary system clock. To enable this feature, enter the command: **vboxmanage modifyvm {your_guest_name} --hpet on**. This command only needs to be issued once and will take effect when the Guest is started. The HPET clock is preferred by OpenVMS, but the system will select another clock source (typically LAPIC) if HPET is not available or enabled.

Now start your Guest by pressing the green arrow labeled **START**. If you have configured your Guest correctly, the console window should appear, showing the UEFI file system devices and a **Shell>** prompt.

If the UEFI Shell does not appear when you start your Guest, review the previous setup steps or contact VSI Support for assistance.

DVD INSTALLATION: If you are installing the EAK from a DVD then simply starting the Guest should launch the OpenVMS Boot Manager.

NETWORK INSTALLATION: The VMS_KITBOOT utility is used to start the network installation procedure. VMS_KITBOOT is launched from your UEFI Shell> prompt.

If VMS_KITBOOT is not present, double check your Storage setup and the Virtual Disk selection. If you copied the Virtual Disk image file using FTP, did you remember to select BINARY transfer mode? If VMS_KITBOOT runs but is unable to locate a suitable network connection, double check your Network setup and make sure the Host is online and able to forward requests from the Guest.

Chapter 20. KVM Guest

Creating a KVM Guest

The following examples illustrate how VSI engineers use the KVM Virtual Machine Host running under CentOS. Other combinations of Host operating systems and remote connections are certainly possible. These examples are intended to provide just enough information for you to work out any details specific to your configuration.

Note

The helpful scripts provided on VSI's FTP site supercede information in this document.

The general idea is to create TWO remote connections to the Host system, one for managing the Virtual Machine and the other for user connection to the Virtual Machine Guest running OpenVMS. The provided script is intended to simplify the creation of a KVM Guest suitable for OpenVMS.

Step 1: Set up a Terminal Server

To support connections from terminal emulators, you need to be running a VNC Server (or similar Terminal Server) on your host system. This example uses the popular VNC Server.

VNC Server Configuration:

1. Run **vncpasswd** to set up the VNC config directory and assign a password for your VNC Server

2. In your **$HOME/.vnc** directory, set up a VNC startup script named: **xstartup**

For example:

```
$ cat xstartup
#!/bin/sh
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRES
exec /etc/X11/xinit/xinitrc
```

To start a VNC Server, login and issue the following command:

 # vncserver &

On the vncserver command line, you can also specify a port, such as: **vncserver :3** VNC Ports are based on number 5900, so specifying 3 becomes port 5903. You will get an error message if that port is already in use, otherwise, it becomes your server port.

If several people will be using the system, it would be useful to assign port numbers to individual users.

Find your vncserver port by issuing the following command:

```
# ps auxwww | grep joe

joe   7106  0.1  0.1 377660 47220 ? Sl Jun20 32:28 /usr/bin/Xvnc :3 .....
```

The *Xvnc :3* indicates your port number is 3.

Step 2: Open Two Terminal Connections

In this example, we will connect a terminal emulator to our VNC Server from an Apple MAC.

Open TWO terminal windows on your MAC (or whatever client system you are connecting from).

Here, the steps can vary depending on your environment. If you have a VNC Client available, it can connect to the graphical Host desktop VNC Server session.

You can also use SSH for the connection as follows:

In the first window, establish a SSH connection to the Host system using your user name, replacing x with your port number, and log in.

$ ssh -L 5901:localhost:590x joe@10.10.10.22 (replace x with your port number followed by your username and IP Address).

In the second window start the GUI.

$ open vnc://localhost:5901

A window will appear asking you to login after which you will get the standard management GUI for the Virtual Machine Manager.

You will always need this two-terminal window setup. If your VNC Server should stop, just start it again.

You can now start the Virtual Machine Manager using either of these two methods:

1. Click on **Applications->System Tools->Virtual Machine Manager**
2. Click on **Applications->Favorites->Terminal** then typing **virt-manager** in the terminal.

Step 3: Create the Guest

Back on the first terminal window, execute the **make-kvm** script provided below. This will create a new Guest which should appear in the GUI's *User Session* list.

In the GUI, open the Guest window and you should be running at the shell.

Also in the first terminal window, connect to the console port of your Guest with the command:

 # virsh console "vm:name"

At this point you should be able to issue commands to the UEFI Shell>

If you are installing the OpenVMS V9.0 EAK from a network and Web Server, issue the command:

Shell> **vms_kitboot**

If you are booting a previously installed OpenVMS Guest, issue the command:

Shell> **vms_bootmgr**

MAKE-KVM Script

Note

The helpful scripts provided on VSI's FTP site supercede any information in this guide. You can download the scripts from the VSI FTP site and edit them to suit your needs. Like all things in the Virtual Machine universe, some fine-tuning may be required.

The following script has been provided as a means of simplifying KVM Guest configuration for OpenVMS. To make the script available to all users, the script should be copied to your Host system in: **/usr/local/bin**

Usage:

make-kvm VMname NumCpus MemSize-in-MBytes

Typical: For the OpenVMS V9.0 EAK, a typical Guest would be configured as a single processor and at least 8GB of memory:

make-kvm VMS_Guest 1 8196

This can be executed as a regular user (no root privileges needed). Some rudimentary parameter validation is done, along with some behind-the-scenes setup for virtual disk backing-store.

The script will create a user KVM Guest. The Guest configuration files will exist in: */home/username/.config/libvirt/qemu/*.xml*

The virtual disk backing-store will be created/maintained in */vm_storage/users/username/images*

Upon successful completion of the make-kvm script, the Guest should be up-and-running and sitting at the UEFI Shell prompt.

Any advanced configuration needs are left as an exercise for the user.

Example:

```
$ make-kvm VMS_Guest 1 8196

WARNING  Disk /usr/share/OVMF/UefiShell.iso is already in use by other guests.
WARNING  No operating system detected, Virtual Machine performance may suffer.
Specify an OS with --os-variant for optimal results.

Starting install...
Domain creation completed.
joe KVM guest VMS_Guest successfully created.
```

Note

About Snapshots:

Most Virtual Machine Hosts provide a means of taking a *snapshot* of the current Guest execution state. This allows you to reload and resume execution from a previously saved snapshot. However, it appears that KVM does not currently support snapshots when a Guest has been configured to use UEFI. Because OpenVMS depends on UEFI, KVM snapshots are not available.

The following script is available for download from the VSI FTP site. It should be copied to your Virtual Machine Host at: /usr/local/bin/make-kvm

```
#!/bin/bash
#
# Usage make-kvm VMname NumCpus Memsize-in-MB
#
# Creates a basic KVM guest for a user session. User specifies
# number of guest CPUs required and the guest memory size. The
# UEFI Shell is provided to the guest, along with a virtual
# disk that has an EFI partition containing vms_kitboot.efi

# Sets up a user storage area for user's KVM guest if one doesn't
# exist.
#
if [ $# -lt 3 ];
then
 echo "Usage: $(basename $0) VMname NumCpus MemSize-in-MB"
exit 1
fi
```

```
VMname=$1
declare -i vCpus=$2
declare -i vMem=$3

# Parameter validation

if [ ${vCpus} -le 0 ];
then
 echo "Invalid value for NumCpus; must be > 0"
exit 1
fi

if [ ${vMem} -le 0 ];
then
 echo "Invalid value for MemSize; must be > 0"
exit 1
fi

virsh list --name --all | egrep "^${VMname}$" >/dev/null 2>&
if [ $? -eq 0 ];
then
 echo "User ${USER} KVM guest ${VMname} exists"
exit 0
fi

# Check if user storage area is set up. If not, set it up here.

TemplateDisk=/vm_storage/shared/images/vms_efi_partition.qcow2
UserDir=/vm_storage/users/${USER}/images
UserDisk=${UserDir}/${VMname}.qcow2

if [ ! -d ${UserDir} ];
then
 mkdir -p ${UserDir}
 if [ $? -ne 0 ];
 then
  echo "Unable to create user storage area ${UserDir}"
 exit 1
 fi
fi

# Copy the template EFI partition disk with vms_kitboot if it doesn't exist

if [ ! -r ${UserDisk} ];
then
 cp ${TemplateDisk} ${UserDisk}
 if [ $? -ne 0 ];
 then
  echo "Unable to copy template disk to ${UserDir}"
  exit 1
 fi
fi

# Validation and setup complete. Create the KVM guest.

virt-install \
--connect qemu:///session \
```

```
--name ${VMname} \
--memory ${vMem} \
--vcpus ${vCpus} \
--cpu host \
--hvm \
--virt-type kvm \
--arch x86_64 \
--machine pc-q35-rhel7.6.0 \
--features acpi=on,apic=on,pae=on,vmport=off,smm=on \
--disk ${UserDisk},bus=sata \
--disk /usr/share/OVMF/UefiShell.iso,device=cdrom,bus=sata \
--check path_in_use=off \
--boot loader=/usr/share/OVMF/OVMF_CODE.secboot.fd,loader_ro=yes, \
loader_secure=no,loader_type=pflash \
--clock tsc_present=yes,hpet_present=yes \
--wait 0 \
--graphics spice

if [ $? -eq 0 ];
then
 echo "${USER} KVM guest ${VMname} successfully created."
 exit 0
 else
  echo "${USER} KVM guest ${VMname} error encountered during creation"
  exit 1
fi
```

Chapter 21. Transferring files between Guest and Itanium Systems

The best way to transfer files between your Virtual Machine Guest and OpenVMS Itanium systems is to use FTP. The V9.0-D release contains TCP network support, including FTP. Refer to the chapter titled *Network Examples* for example network configurations and using FTP.

If you prefer to use physical media to transfer files, we recommend creating a small *Transfer Disk* that can be quickly copied between your host and your OpenVMS Itanium development system. A few command procedures can make the task less painful.

Creating and Accessing a Transfer Disk

STEP 1: Create the Disk on an OpenVMS Itanium System

On your OpenVMS Itanium system, use the LD Utility to create a raw .DSK (disk image) container file.

If the LD Utility has not been started, execute $ @SYS$STARTUP:LD$STARTUP.COM.

The following command procedures illustrate how to create a new transfer disk and then how to access the disk when moving it back and forth to your Virtual Machine.

```
$! EXAMPLE 1: Create a new 100MB Transfer Disk
$!
$! Adjust the size as needed (512 byte blocks)
$! Blocks = (MB * 1024 * 1024) / 512
$ LD CREATE TRANSFER.DSK /SIZE=204800
$ LD CONNECT TRANSFER.DSK /SYMBOL
$ INITIALIZE LDA'ld_unit': TRANSFER
$ MOUNT/OVER=ID LDA'ld_unit':
$!
$! Replace items in {braces}
$ CREATE/DIRECTORY LDA'ld_unit':[{MYDIRECTORY}]
$! Create any other directories you need
$!
$ COPY {MYFILES.EXT} LDA'ld_unit':[{MYDIRECTORY}]
$! Copy any other files you need
$!
$ DISMOUNT LDA'ld_unit':
$ LD DISCONNECT LDA'ld_unit':
$ EXIT


$! EXAMPLE 2: Access an existing Transfer Disk
$!
$ LD CONNECT TRANSFER.DSK /SYMBOL
$ MOUNT/OVER=ID LDA'ld_unit':
$!
$! Replace items in {braces}
$ COPY {MYFILES.EXT} LDA'ld_unit':[{MYDIRECTORY}]
$! Copy any other files you need
$!
$ DISMOUNT LDA'ld_unit':
$ LD DISCONNECT LDA'ld_unit':
```

```
$ EXIT
```

STEP 2: Transfer the Raw Disk to your Guest

Using FTP (in binary mode), transfer the raw disk container file to your Virtual Machine Guest. We highly recommend using the *FileZilla* FTP transfer utility if your guest is hosted by Windows or Apple OS. It works very well with OpenVMS, but any suitable transfer utility will do the job.

You can PUT or GET the file, depending on which side your FTP is on. This example assumes you are pushing the file to your Guest from OpenVMS.

$ **FTP**

FTP> **OPEN {your Guest system}**

FTP> **CD {your Guest folder}**

FTP> **BINARY**

FTP> **PUT TRANSFER.DSK**

FTP> **QUIT**

On your Guest system, rename the file to remove the semicolon and version number.

STEP 3: Convert your raw disk to a virtual disk

This example shows conversion to a VirtualBox VMDK disk. To convert to a KVM QCOW2 disk, refer to the chapter titled *Virtual Disk Format Conversions.*

In a terminal command window on your Guest system, issue the following command to convert your raw disk into a VMDK.

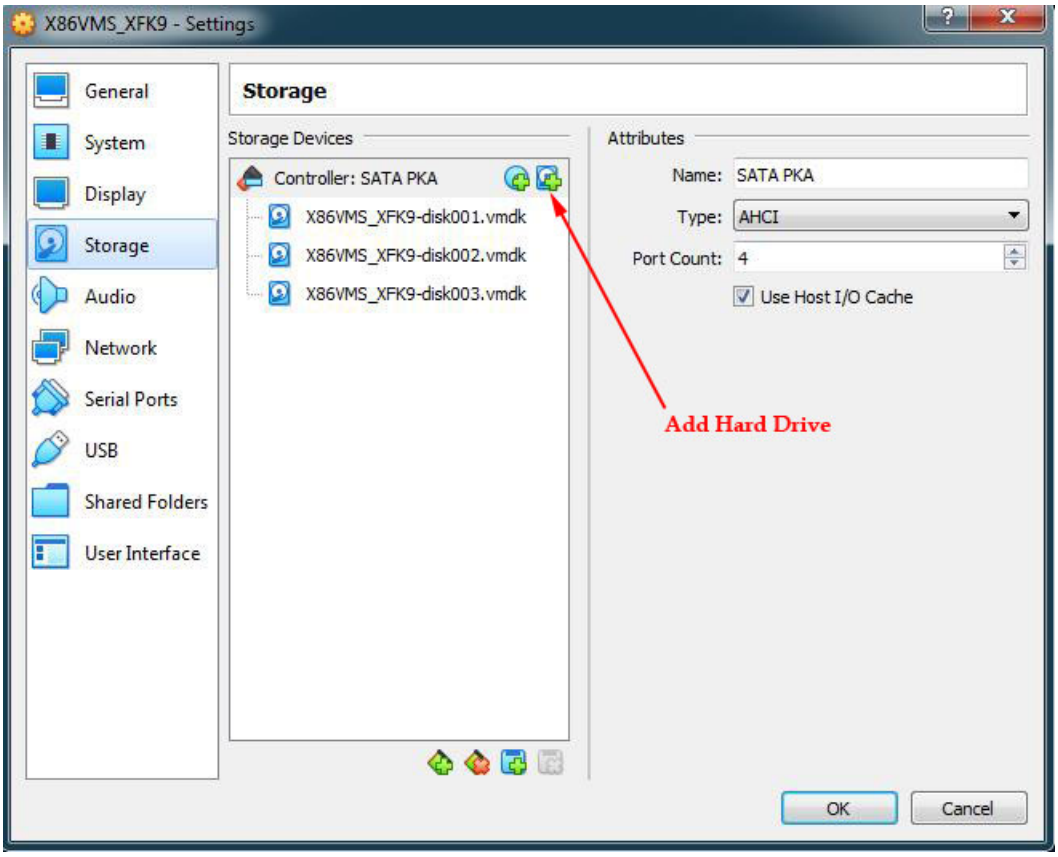
vboxmanage internalcommands createrawvmdk -filename target.vmdk -rawdisk target.dsk

The conversion should only take a few seconds. When done, you will have both the original .DSK file and a new .VMDK (fixed size) file. It is important to keep these two files together. The .VMDK file is a simple ASCII description of the disk image, while the actual contents remain in the .DSK file. This is important, because even after you connect the .VMDK virtual disk to your Guest, you can still copy the .DSK file back to your OpenVMS Itanium and access it as shown previously to copy files.

STEP 4: Connect your raw disk to your Guest

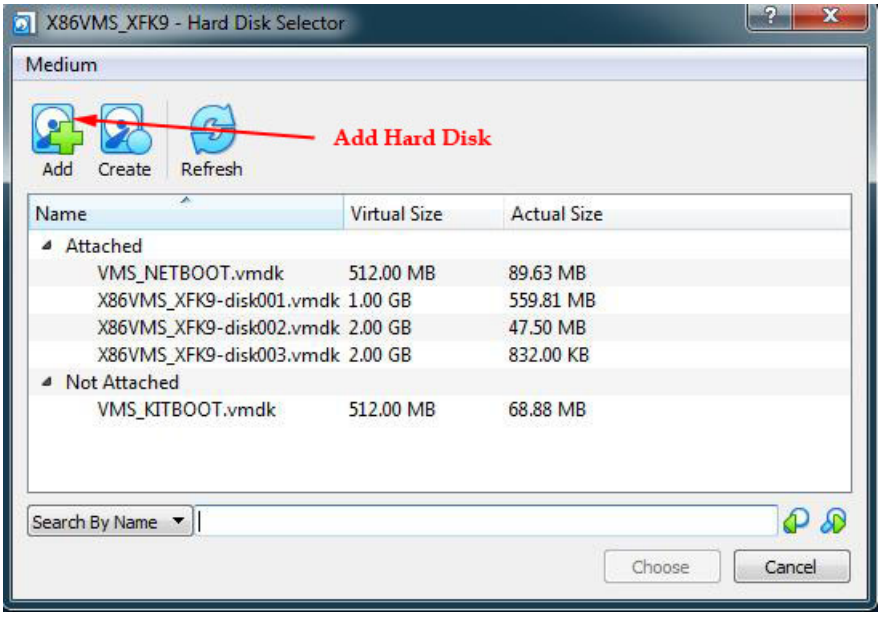
1. If you don't already have one, add a SATA Controller to your Guest Storage using the VirtualBox GUI. By connecting your new transfer disk as a SATA Hard Disk, it will appear as a DK device under OpenVMS.

2. On the line showing the SATA Controller and press the *Add hard drive* icon to display a *Hard Disk Selector* dialog box.



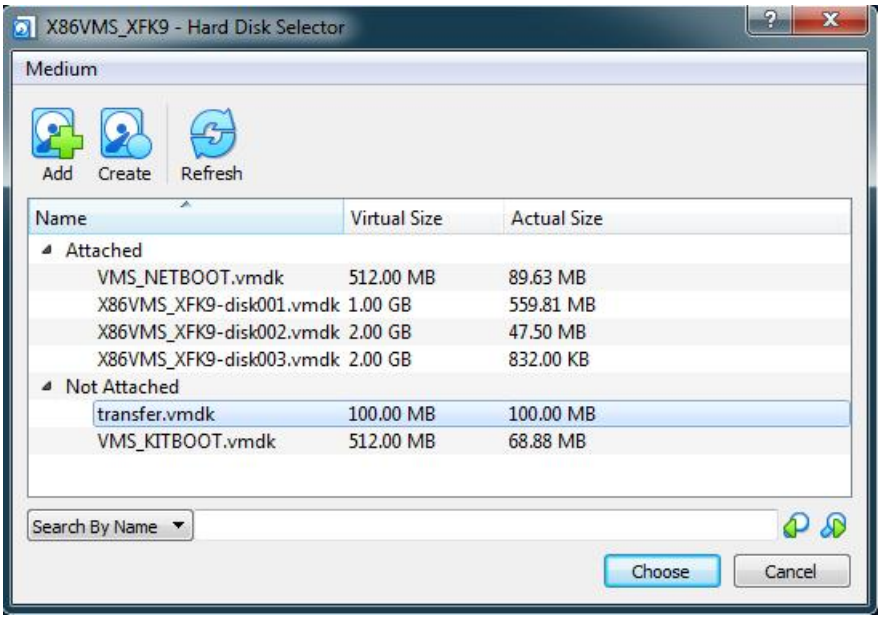
Adding a SATA Hard Drive

3. In the Hard Disk Selector dialog box, press the *Add* button and select your disk image file and then press *Open*. This makes the new drive available to guests.



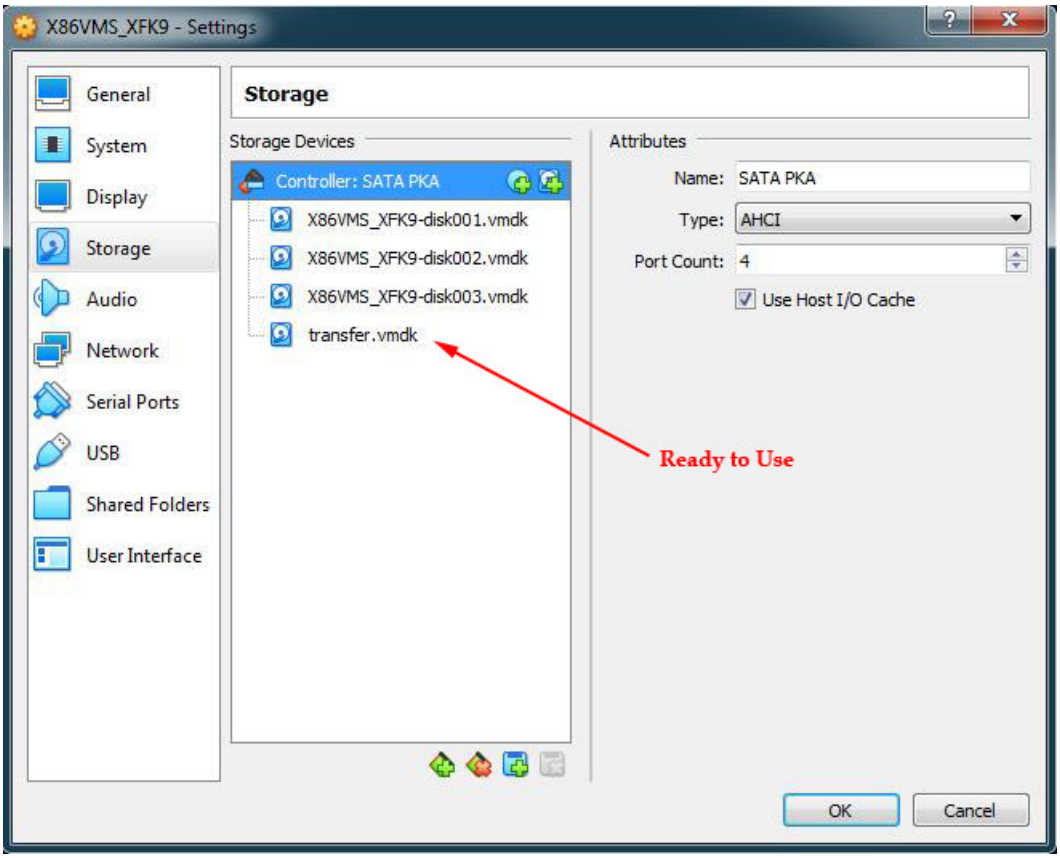
Adding your Transfer Disk

4. Select the entry you just added and press *Choose*.



Connecting your Transfer Disk

5. Your new Transfer disk is ready to use.



Ready to use!

When you boot your Guest, your new transfer disk will show up as a DK disk (i.e. DKA100). You can mount it as you would with any OpenVMS disk and access your files.

When you need to move files back to your OpenVMS Itanium system:

1. Shut down your Guest.

2. FTP the .DSK file back to your Itanium system.

3. Access the .DSK file using the command procedure shown previously.

4. FTP the .DSK file back to your Guest folder where it previously resided.

5. Start your Guest and mount the device again.

Note that this procedure only works with fixed-size VMDK disks. If the disk is altered in any way that makes it incompatible with the description in its associated .VMDK file, then you will get an error when you attempt to use the .DSK again. In this case, you can simply issue the vboxmanage command again to convert the raw .DSK file to a new .VMDK (using a different name) or if you would like to keep the same name, use the VirtualBox Media Manager to remove the previous disk from your Guest, then do the conversion again. This is necessary because VirtualBox uses GUIDs to uniquely identify disk images and associate their names.

Part VI. Network Examples

Configuring the Network

Table of Contents

Chapter 22. Useful Network Configuration Examples

This chapter contains configuration examples for networking between your Guest and a remote server or between two Guests on a local network.

The V9.0-D release contains a preliminary release of TCPIP along with Telnet and FTP utilities. Refer to the appropriate documentation regarding details of configuring your network.

NAT vs Bridged

One of the first questions that comes up when configuring the network is whether to define a Guest network adapter as a *NAT* or *Bridged* adapter?

Think of it this way, NAT gets you to the internet, Bridged devices get you to the local network.

In NAT mode, the Guest network interface is assigned by default to the IPv4 range 10.0.x.0/24 where x corresponds to the instance of the NAT interface+2. So x equals 2 when there is only one NAT instance active.

In this case, the Guest is assigned to the address 10.0.2.15, the Gateway is set to 10.0.2.2 and the Name Server can be found at 10.0.2.3

If the NAT network needs to be changed, use the following command:

$ VBoxManage modifyvm VM-name --natnet1 "192.168/16"

This command reserves the network addresses from 192.168.0.0 to 192.168.254.254 for the first NAT network instance of VM-name. The guest IP would be assigned to 192.168.0.15 and the default gateway could be found at 192.168.0.2

All VMs can use the same NAT network. But if you added multiple NAT networks, then for the second VM, the addresses are 10.0.3.x. For the nth VM, 10.0.1+n.x

Special Notes for NAT Operation

The VirtualBox VM for NAT emulates a router with these implied addresses:

IP Address	Usage
10.0.2.1	VBOX Router
10.0.2.2	Default gateway
10.0.2.3	DNS Server #1
10.0.2.4	DNS Server #2
10.0.2.5	DNS Server #3
10.0.2.6	DNS Server #4
10.0.2.15/24	EIA0 assigned IP address
127.0.0.1	EIA0 loopback interface

VirtualBox defines these addresses for the first VM defined.

For the second VM, the addresses are 10.0.3.x. For the nth VM, 10.0.1+n.x.

NAT Addresses used by VirtualBox

Network Example 1: Quick Example

One example of a handy network configuration is to set up two Virtual Machine Guests, each with their first NIC (EIA0) defined as NAT (internet access), and their second NIC (EIB0) defined as Bridged (local network access).

Both Guests will use the same address for EIA0 – 10.0.2.15/24 and their default-route is set to 10.0.2.2

For EIB0, one Guest will use the address of a Windows-10 PC's NIC + 1, and the other VM will use the address of NIC + 2.

With this configuration we can:

1. FTP to our remote lab systems (via NAT) from both VMs (simultaneously)

2. Ping both the local Windows-10 PC and my remote lab systems

3. Ping and FTP one Guest from the other. When accessing the other Guest, I use the EIB0 address.

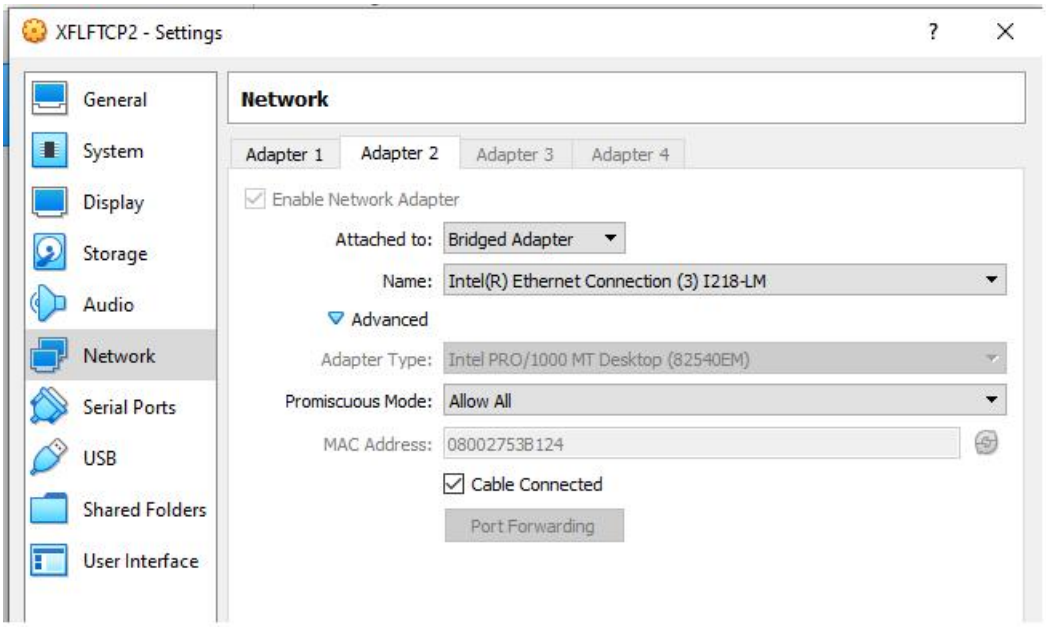
Note that we used numeric addresses, so set DNS to "NO".

The next section shows this example with some useful pictures.

Network Example 2: Step by Step.

Here is the same example, but in steps and pictures...

Step 1: Configure a 2nd NIC in VBOX Settings. Set it to *Bridged* networking and set the Advanced Settings as shown:



Second NIC Settings

Step 2: Check with your home network manager or management utility to get an assigned IP address. On a Windows PC, you can use the **IPCONFIG** command. Add 1 to the IP address of the system and ping that address to see if it was free.

Step 3: Boot VMS but *before starting the network via @sys$startup:ip$startup* do the following command:

$ ip config

Add the new interface (se1) and configure the 2nd NIC as EIB0 using the IP address and subnet mask so you end up with:

```
$ ip config

VSI TCP/IP Network Configuration Utility

NET-CONFIG> show

Interface                                Adapter  CSR Address  Flags/Vector
---------                                -------  -----------  ------------
se0      (Shared VMS Ethernet/FDDI)       -NONE-     -NONE-     -NONE-
 [TCP/IP: 10.0.2.15, IP-SubNet: 255.255.255.0 (/24),IP-Broadcast: 10.0.2.255]
 [VMS Device: EIA0, Link Level: Ethernet ]

se1      (Shared VMS Ethernet/FDDI)       -NONE-     -NONE-     -NONE-
 [TCP/IP: 192.168.1.26,IP-SubNet:255.255.255.0(/24),IP-Broadcast: 192.168.1.25
 [VMS Device: EIB0, Link Level: Ethernet ]

   Official Host Name:             myguest.home.mycompany.com
   Default IP Route:               10.0.2.2
   Timezone:                       EST
   Timezone Rules:                 US/EASTERN
   Load UCX $QIO driver:           TRUE
   Load PWIP (Pathworks) driver:   TRUE
```

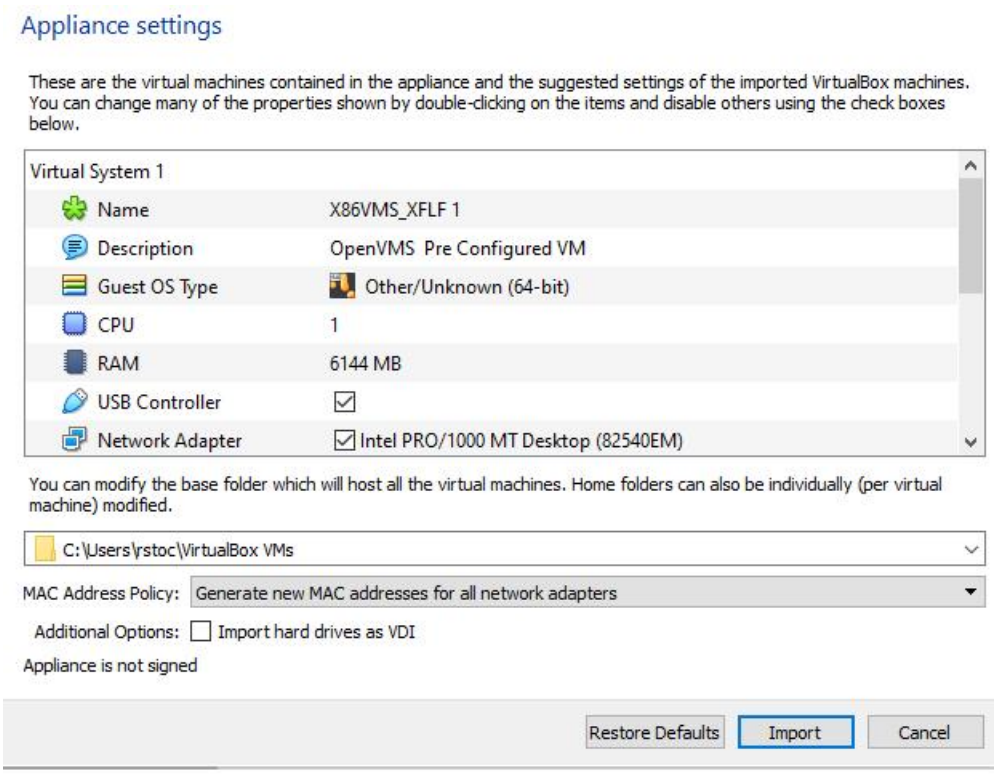
Network Example 3: From Appliance Import through FTP

Here is a complete example starting from Virtual Appliance import:

Step 1: In VirtualBox Manager, select File then Import Appliance:

Import the OpenVMS Virtual Appliance

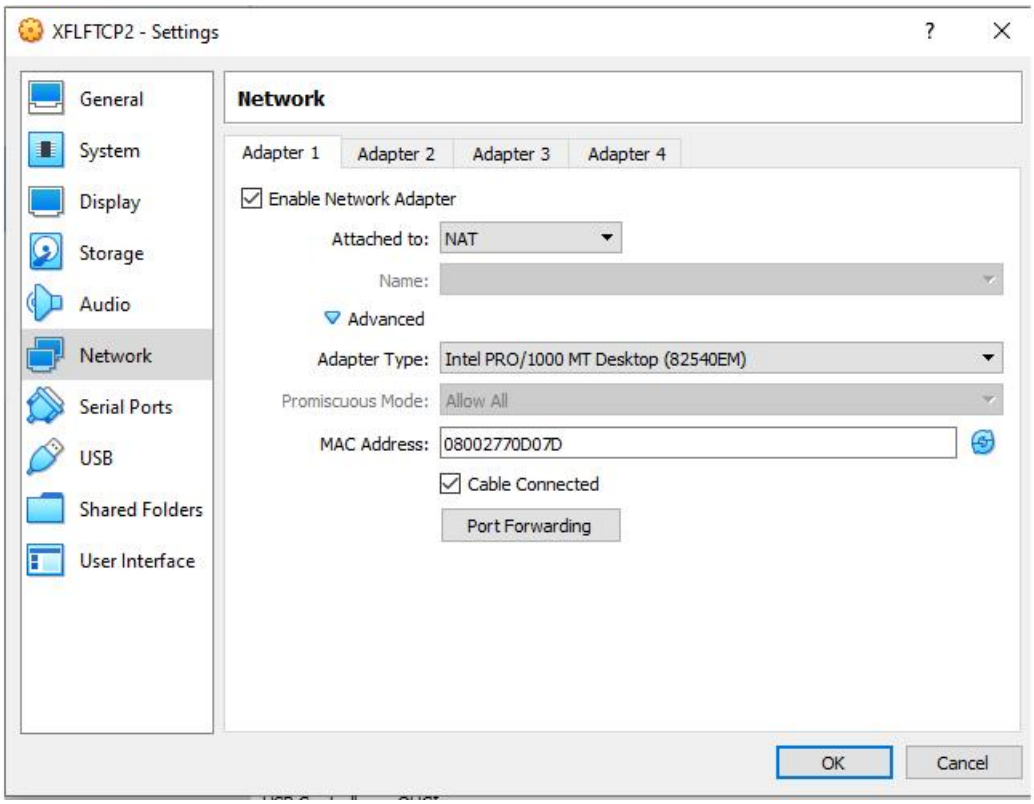
Step 2: Select Next, then select *Generate new MAC addresses*, otherwise, if you have multiple Guests with the same addresses you might have issues. Uncheck the box *Import hard drives as VDI*. Then press *Import*.



Import Option Selection

Step 3: Select *Settings* and *Serial Port* and verify that the TCP/IP port numbers are what you want for each port.

Step 4: Select *Network* and ensure the NIC is set to *NAT*.



NIC NAT Settings

Step 5: Start the Guest, connect the PuTTY (or similar) session to the serial port, then BOOT.

Step 6: Once booted, login and do:

```
$ @ip$:configure

VSI TCP/IP Initial Configuration

%IP-W-INTDEFINED, interfaces have already been defined

Changes to your existing interfaces should be made with the
IP CONFIGURE /NETWORK command.

You can replace your existing interface configuration now.

Do you want to replace your existing interface configuration? [NO] yes

Enter your fully-qualified Internet host name : myguest.home.mycompany.com

  VMS                 TCP/IP
  Device  Link  Speed  Interface  Address          Subnet           IPCI
  ------  ----  -----  ---------  ---------------  ---------------  ----
  EIA0    Up     1000                                               no

Select device to configure or enter NONE [EIA0] :

Enter the Internet (IP) Address for interface EIA0 : 10.0.2.15

Enter the Subnet mask for interface EIA0 : 255.255.255.0

Enter the IP Address of your default route : 10.0.2.2

Do you want to use DNS to resolve host names [YES] ? no

Enter your local timezone : est

  Interface:  EIA0
  Host Name:  myguest.home.mycompany.com
  IP Address:  10.0.2.15
  Subnet Mask:  255.255.255.0
  Default Route:  10.0.2.2
  Use DNS:  no
  Run DNS Server:  no
  Time Zone:  EST

Are you satisfied with these values? [YES]

Configuration saved
$
$ @sys$startup:ip$startup
```

Step 7: Test Ping and FTP:

```
$ ip ping 10.10.100.7
   PING 10.10.100.7 (10.10.100.7): 56 data bytes

   64 bytes from 10.10.100.7: icmp_seq=0 time=49 ms
```

```
   64 bytes from 10.10.100.7: icmp_seq=1 time=29 ms

$ [ftp://ftp 10.10.100.7/user=myusername]ftp 10.10.100.7/user=myusername

  myguest.home.mycompany.com VSI TCP/IP FTP user process V5.5(121)
  Connection opened (Assuming 8-bit connections)

    FTP Server (Version 5.7) Ready.

  Username: xxxxx
  Password: yyyyy

    User logged in.

  FTP> get xxc.com

    To local file:
    Opening data connection for XXC.COM (10.9.1.19,62957) (74 bytes)
    Transfer complete.

  FTP>
```

Part VII. Crash Dump Kernel

A new way of handling crashes

Table of Contents

Chapter 23. The Dump Kernel

As an Early Adopter, there is an elevated chance of encountering a Bugcheck and crash. OpenVMS x86 implements a new way of quickly saving memory contents for subsequent analysis and returning the system to service. Crash processing is performed using a second instance of the operating system kernel, known as the *Dump Kernel.*

Dump Kernel Overview

The Dump Kernel is a limited-functionality OpenVMS Kernel designed to handle crash dump data collection, compression and storage.

During boot, the Dump Kernel is created in parallel with the Primary Kernel. It resides in a separate memory space and shares the MemoryDisk. The Dump Kernel requires only 512MB of memory.

The MemoryDisk remains in memory after the Primary Kernel is booted and the device associated with the MemoryDisk (DMM0) is set offline and write-protected. A logical disk device (LDM1) is created during boot, to allow access to files residing in the MemoryDisk using symlinks.

Having a second kernel for crash dump processing reduces the dependency on the crashing system and allows greater flexibility in collecting, compressing and writing to the dump device/s using runtime drivers. OpenVMS x86 does not use the traditional Boot Drivers.

The Dump Kernel is loaded, but does not boot until the Primary Kernel encounters a Bugcheck. The Bugcheck process prepares the crashing system (halting processors, identifying important areas of memory, etc.), then transfers control to the Dump Kernel's SYSBOOT image and the Dump Kernel boots.

The Dump Kernel, being resident in memory, boots quickly. A Dump-Kernel-specific startup procedure: [SYSMGR]SYS$DUMP_STARTUP.COM executes the Dump Kernel's main image: [SYSEXE]SYS$DUMP_KERNEL.EXE. The Dump Kernel maps and collects the important areas of memory, compresses and writes the data to the dump file: SYS$DUMP.DMP for later analysis.

Once the crash dump has been processed and the dump file has been written, the Dump Kernel will invoke a power off or system reset to restart the Primary Kernel and re-arm a new instance of the Dump Kernel.

Note

Process Dumps are not yet supported by the EAK. This will be added in a future release.

The Dump Kernel has a few new system parameters to control its operation. These are described in following sections.

Preparing a Dump Device

To create a Dump File on your System Disk, use **SYS$UPDATE:SWAPFILES.COM**. When the command procedure requests a file size, start with 200000 blocks (extend as needed).

Note

You can ignore the message about rebooting. If the Dump File exists, it will be used.

To create a Dump File on a different disk, use the following procedure. We will assume DKA100 in this example and using System Root 0 (SYS0):

$ **CREATE/DIR DKA100:[SYS0.SYSEXE]**

$ **RUN SYS$SYSTEM:SYSGEN**

SYSGEN> **CREATE DKA100:[SYS0.SYSEXE]SYSDUMP.DMP/SIZE=200000**

SYSGEN> **EXIT**

$

Refer to the next section to add your dump device to the list of available devices.

Specifying one or more Dump Devices

Crash Dumps can be written to the System Disk or to one or more designated dump devices. In prior releases of OpenVMS, designating dump devices was accomplished through UEFI Environment Variables. For OpenVMS x86, we have chosen to minimize the use of Environment Variables. Instead, specify your dump device by editing the file: **SYS$MANAGER:SYS$DUMP_CONFIG.DAT**.

$ **EDIT/EDT SYS$MANAGER:SYS$DUMP_CONFIG.DAT**

Device=DKA100:

CTRL-Z

EXIT

$

Use a comma-separated list to specify additional dump devices: *Device=ddcu:,ddcu:,ddcu:*

You must reboot the system after changing Dump device parameters.

Note

At the time of this writing, there is a plan to add new options for the *SET* command to manage dump device entries and eliminate the need to reboot, but this is not yet available for the EAK release.

Control Parameters

The SYSGEN parameter **DUMPSTYLE** must be set. OpenVMS V9.0 supports only *selective* (bit 0) and *compressed* bit (3) dump features. Setting DUMPSTYLE to 9 enables these features for a dump to the System disk. To enable dumping to a non-System disk include setting bit 2 by setting DUMPSTYLE to 13 (decimal).

$ **RUN SYS$SYSTEM:SYSGEN**

USE CURRENT

SET DUMPSTYLE 13

WRITE CURRENT

EXIT

$

You must reboot the system after changing Dump control parameters.

Automatic Actions

When the Dump Kernel completes its tasks, it will execute a user-specified automatic action to restart or power-off the system. Several parameters are involved in defining this behavior. The following is a description of how these parameters interact.

The flags **OPC$REBOOT** and **OPC$POWER_OFF** are set or cleared in response to SHUTDOWN parameters (or answers to SHUTDOWN prompts).

1. If **OPC$REBOOT** is set at shutdown (Operator Requested Shutdown is a form of Bugcheck) or SYSGEN parameter **BUGREBOOT** is set during any other form of Bugcheck, the system will be RESET and will return to the UEFI Shell> prompt.

   Be aware that any automatic actions established for UEFI and the Boot Manager will then occur.

   Automatic reboot is achieved by invoking VMS_BOOTMGR in startup.nsh and setting AUTO BOOT in VMS_BOOTMGR.

2. If **OPC$POWER_OFF** is set at shutdown, the system or Virtual Guest will be powered off.

3. If both **OPC$REBOOT** and **OPC$POWER_OFF** are set (an illogical combination), the system or Virtual Guest will be reset and any automatic actions will be taken.

4. If neither **OPC$REBOOT** or **OPC$POWER_OFF** are set at shutdown and the SYSGEN parameter **BUGREBOOT** is NOT set when any other Bugcheck occurs, the Operator will be prompted to press any key to RESET the system and return to the UEFI Shell>. In other words, with no automatic action specified, the system will wait for an Operator. The Operator can then decide whether to allow or abort any subsequent automatic actions from UEFI (startup.nsh) and the Boot Manager (AUTO BOOT).

   This of course, assumes that a terminal session on the COM 1 port is available for keyboard input.

Additional failure modes are handled by performing a system RESET to the UEFI Shell> and any specified automatic actions that follow. These failure modes include:

- The Primary Kernel fails to start the Dump Kernel.
- The Dump Kernel itself, Bugchecks.
- The Dump Kernel image fails and OPCCRASH gets started.

Crash analysis is beyond the scope of this guide. The traditional OpenVMS crash analysis tools are available. The command: $ **ANALYZE/CRASH {dumpfile}** is a good starting point.

Part VIII. Troubleshooting

Helpful Tips and Tools

Table of Contents

Chapter 24. Troubleshooting

Tools of the Trade

The OpenVMS V9.0 Early Adopters Kit is not a complete production kit. For one, it lacks a User Code Symbolic Debugger and other useful development tools. Instruction level debugging is available using *XDELTA* or *DELTA*debuggers and code listings. Being a pre-production kit, many other features are available to enable extended messages, run diagnostic tests, etc.

The following are just a few troubleshooting tools. Contact VSI for any assistance you may need.

XDELTA

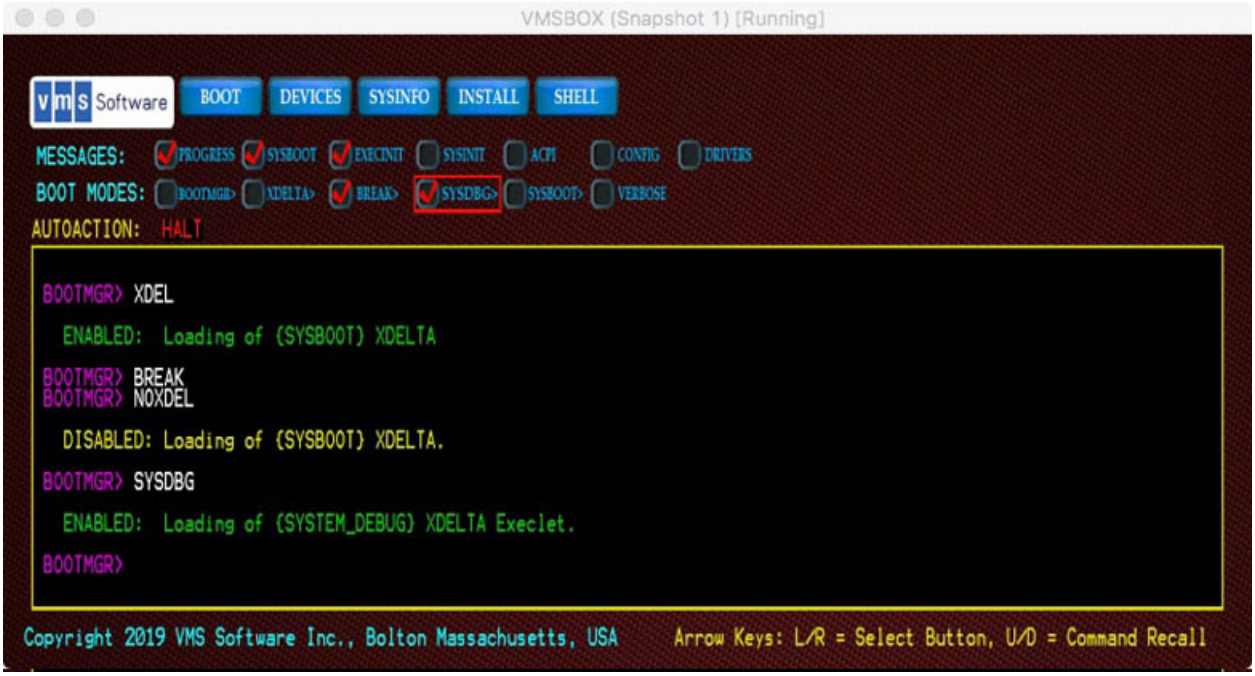
The venerable XDELTA debugger is available and has been taught some new tricks which may be helpful to early developers. A XDELTA User Guide is available. VSI has implemented several XDELTA extensions that can help with Page Table related issues among other things. These extensions are undocumented, but available through VSI as-needed.

XDELTA comes in two flavors:

XDELTA is compiled into the primary bootstrap program SYSBOOT.EXE for use during the early boot stages where the system is transitioning from physical to virtual addressing. To invoke this early XDELTA, issue the BOOTMGR> **XDELTA and BREAK** commands prior to booting. This will enable the debugger and take the first available breakpoint inside SYSBOOT.EXE.

XDELTA also exists and a loadable executive image. This allows it to be used on programs that run in virtual address space including drivers. To invoke the loadable XDELTA, issue the BOOTMGR> **SYSDBG and BREAK** commands prior to booting. This will load the debugger and take the first available breakpoint inside EXEC_INIT.EXE. If you prefer to take breakpoints inside program modules, insert **ini$brk()** calls in your sources.

Refer to the VSI XDELTA User Guide for usage details.



Booting with XDELTA

The above screen shot shows the enabling of early XDELTA, then disabling it and enabling the *loadable XDELTA* Instruction-level Debugger.

Boot Manager Startup

Note

Some of the Boot Manager diagnostic features will be removed for production releases.

The Boot Manager provides extensive diagnostic messages to help troubleshoot any problems that affect the loading and initialization of the system prior to transfer to SYSBOOT.EXE. Once the first SYSBOOT message appears, the Boot Manager's job is over and system debugging tools are required for further problem analysis.

Setting the Boot Manager Interaction flag: BOOTMGR> **BOOTMGR** enables messages with details of the early boot process up to the transfer of control to SYSBOOT.EXE. Along with additional messages, the user will be prompted for examination of key data structures related to boot.

In rare cases where the Boot Manager has trouble relocating images, setting the **BOOTMGR and VERBOSE** flags together will display intricate details of image relocation, but aside from a mild therapeutic value for insomniacs, this takes a long time and is best avoided.

The Boot Manager evaluates and allocates memory very early after being launched and again just before transfer of control to SYSBOOT.EXE when the memory map is finalized. To observe further details pertaining to memory mapping, issue the command: BOOTMGR> **MEMCHECK** prior to booting. Use this diagnostic in conjunction with the BOOTMGR> **HALT** flag and BOOTMGR> **RESET** command.

The Boot Manager evaluates potential boot devices very early after being launched and again when the system is being booted. To observe further details pertaining to device discovery and enumeration, issue the command: BOOTMGR> **DEVCHECK**. Additional device related information can be obtained from the BOOTMGR> **PCI, USB, and DEVICE LIST** commands.

Logging of Boot Manager messages must be done through a connected terminal emulator. The V9.0 Boot Manager does not itself provide logging or scroll-back features.

If you wish to capture a Boot Manager screen shot, you can press the F1 key at any BOOTMGR> or PAGE prompt and a bitmap file named *snapshot.bmp* will be saved to the root UEFI file system device (fs0: etc.). Only one file will be saved at a time and is over-written each time F1 is pressed.

MemoryDisk

The MemoryDisk (SYS$MD.DSK) is a Logical Disk (LD) container file which is critical for booting as it contains all of the files that comprise a minimum bootable OpenVMS Kernel. The structure and custom Boot Block used during boot are critical and should not be tampered with.

The MemoryDisk is initialized and maintained by a system command procedure *SYS$MD.COM*. This procedure is run during installation and again in the event that a file contained in the MemoryDisk is updated through a PCSI Kit or Patch installation. This is done automatically by the various utilities that are concerned with the MemoryDisk. The procedure takes great care to maintain the integrity of the MemoryDisk as well as the symlinks that are used by the OS when referencing these files. Under normal circumstances SYS$MD.COM should never be invoked manually unless instructed to do so by VSI Support for specific troubleshooting reasons.

Because the MemoryDisk itself is a bootable entity, located inside the System Disk and having three partitions of its own, there are several critical boot blocks and related structures involved in booting. Use care to avoid renaming or relocating the files: *SYS$MD.DSK* (the MemoryDisk) or *SYS$EFI.SYS*

(the UEFI Partition) on the System disk. Additionally, avoid adding or removing files from within the MemoryDisk or UEFI Partition. Corruption of either of these container files may result in an unbootable system.

Network/Web Server & VMS_KITBOOT

Many things can interfere with network booting. When you initially launch VMS_KITBOOT.EFI it will attempt to identify available network adapters. Failing to locate a suitable adapter is a clear sign that your Guest network is not properly configured. Verify that you have defined a network adapter and that your Host is able to access the network itself since all network traffic ultimately passes through the Host.

If you Guest adapter appears to be properly configured but still fails to respond, try changing your adapter type from *Bridged Adapter* to *NAT* or vice versa. If this does not resolve the problem, seek assistance from a network administrator or other sources familiar with your chosen Virtual Machine product.

The VMS_KITBOOT.EFI utility is based on the open-source iPXE software. When this underlying software encounters an error, it displays a URL and error code. If you enter this URL and code into a web browser, it will take you to a brief explanation of the error. While these messages are not always detailed, they may point you in the right direction for solving connection issues.

Once VMS_KITBOOT.EFI has selected a network adapter, it attempts to locate a Web Server at the IPV4 address that you provided in response to the prompt. The full URL of this request must coincide with your Web Server setup. For example, if you entered the IPV4 address: 10.10.10.22 then VMS_KITBOOT will fabricate the URL as: http://10.10.10.22/netboot/

This implies that you have a Web Server running at this IP Address, that you have a project folder/directory named **/netboot/** and that the installation files have been copied to this folder/directory and file protection allows for read access. If this file is correctly located and received by VMS_KITBOOT, then the Boot Manager should be downloaded and displayed.

If the kit download fails, make sure you have provided ample memory in your Guest. 8GB is the recommended minimum, but during installation it may help to increase memory. If this solves the problem, you can always reduce memory after installation.

Other things that can cause the network installation to fail are if any of the files were downloaded from the VSI FTP site using the wrong FTP mode (binary versus ASCII). Files of type: .ISO, .EFI and .BMP are BINARY files. Note too that some web-servers do not work with files having OpenVMS version numbers and you will need to rename these files to remove the version number and semicolons. Some web servers are case-sensitive too.

The OpenVMS V9.0 EAK is a large file and may take several minutes to download to your Guest. If VMS_KITBOOT's download percentage counter shows no sign of progress after several seconds, you may need to reset your Guest and try again. If all else fails, you might consider burning the EAK .DSK file to a DVD and trying a local boot from DVD or copy the file to your Guest and boot as a Virtual DVD.

Virtual Machines

Most of the problems we have seen from Virtual Machines come down to user error. Getting a command wrong or failing to define the Guest properly. Review the information provided in the VirtualBox Guest or KVM Guest sections of this document carefully to make sure you have defined your Guest properly.

It is imperative that your Guest presents a UEFI *Shell>* prompt. If you are unable to start a Shell, consult with your Virtual Machine product documentation. Virtual Machines need to run the Shell provided by the product. Although UEFI Shell applications are freely available, their functionality varies widely

and Shells used with Virtual Machines need to be Virtual Machine-aware. Both VirtualBox and KVM support UEFI boot environments but KVM may require installing additional options.

If you cannot determine why the boot is failing, contact VSI Support for assistance.

Problems after Transfer to OpenVMS

If you are able to download and execute the VMS_BOOTMGR.EFI utility but it appears to hang when you issue a BOOT command, try restarting your Guest and setting the various message-related boot flags to get an indication of how far the boot progresses before hanging. The PROGRESS, SYSBOOT and EXEC flags should help.

If you think the Boot Manager has not transferred control to SYSBOOT, set the BOOTMGR flag too. If you see even a single message from SYSBOOT or EXEC_INIT, then you have progressed beyond the Boot Managers role. If it appears that your system hangs when discovering devices, you might set the CONFIG and ACPI boot flags.

The VERBOSE mode flag will enable extended messages in many cases, but be prepared to log a large amount of information. Avoid using the VERBOSE and BOOTMGR flags together.

IMPORTANT: Keep in mind that you MUST use a remote terminal emulator to communicate with OpenVMS. Once you have issued a BOOT command and have executed SYSBOOT, the keyboard becomes unusable. OPA0 messages will continue to be displayed, but the local keyboard will cease to operate as an input device to OpenVMS. You must establish a terminal session to log in and use OpenVMS. Eventually, when VSI has finished porting the various drivers, you will be able to continue using the local OPA0 display and keyboard.

If you cannot determine why the boot is failing, contact VSI Support for assistance.

Chapter 25. Terminal Emulator Tips

The Boot Manager and Kitboot can use terminal emulators and serial ports for input and output. Establishing a remote terminal connection is essential if you intend to use XDELTA and as a general user. The logging feature of your terminal emulator provides the recommended way to obtain a log file of your entire boot process should that be necessary.

The Boot Manager and OpenVMS want control of the terminal characteristics. Most of the issues with setting up a terminal utility involve command modes and line terminations.

For best results, a terminal session should be in *character* mode as opposed to *line* mode. This assures that each character is transmitted as it is entered. This is particularly important when using a debugger since they tend to have single-character command interpreters.

Additionally, avoid having your terminal utility modify line terminators, carriage returns and linefeeds. Let OpenVMS handle the terminators. Lastly, where possible, disable protocol negotiations. A *Telnet* connection generally involves a negotiation sequence unless explicitly disabled, whereas a *Raw* connection has no such issues.

Terminal Connection Sequence

Some terminal emulators will attempt to take ownership of a serial port if it is not already in use. If this happens, the VMS_KITBOOT procedure or the VMS_BOOTMGR utility may consider the port as unavailable and fail to make a connection.

If this occurs, you need to follow a specific sequence when connecting.

1. If using a Virtual Machine, start your Guest and launch the VMS_BOOTMGR.
2. When you reach the BOOTMGR> prompt, start your terminal emulator.
3. Connect using a RAW, TELNET or SSH session.
4. You should now see the BOOTMGR> prompt in your emulator and input should be functional from either session prompt. If this isn't working, try issuing the command: BOOTMGR> **COM 1**. If this still doesn't establish a connection, review the TCP Port you have assigned to COM 1. Refer to the following sections for specific emulator details or contact VSI Support for assistance.

If you would like to route output to a different serial port or to multiple serial ports, enter the command: BOOTMGR> **COM n** (note the space), where 'n' is the x86 architecturally defined serial port from 1 to 4. If you issue this command multiple times for different ports, a list of active ports will be established. The **COM** command with no arguments lists the active ports, and **COM 0** disables all serial port output until OpenVMS is booted, at which time COM 1 (0x3F8) becomes the default OPA0 port.

Using the PuTTY Terminal Emulator

If you wish to direct Console IO to a system running Microsoft Windows, we recommend using the PuTTY terminal emulator. You can download PuTTY from `http://putty.org`. MobaXterm is another excellent terminal utility.

Depending on how the system you are running your terminal emulator on is connected to the target system, you can select a RAW, SSH or TELNET session. The RAW option works well with OpenVMS. If using TELNET, you will need to adjust various settings described below to avoid duplicate output and line termination issues.

PuTTY has several nice features that allow for control of your session. If you are working remotely using a SSH connection through your VPN, make sure that *IP Address Forwarding* is enabled (typically it is by default).

Set up your PuTTY session as follows:

1. Right-click on the window frame of your PuTTY session and select **Change Settings...**. The important settings include the following:

   In *Terminal* settings, check the following boxes:

   - TURN OFF Implicit CR in every LF
   - TURN OFF Implicit LF in every CR
   - TURN OFF Local Echo
   - TURN OFF Local line editing

2. In *Window* settings, set your Columns and Rows to match the Boot Manager display (typically 120 Columns, 40 Rows).

3. If you are using a SERIAL session, use following settings:

   - Speed: 115200
   - Data Bits: 8
   - Stop Bits: 1
   - Parity: None
   - Flow Control: XON/XOFF

If you are using TELNET with XDELTA, you need to put TELNET into CHARACTER MODE in order to avoid having to hit return after every XDELTA command. This can be done via: telnet> **mode char**. telnet> **mode line** returns it to line mode.

Using telnet on Apple macOS

Starting with the MOHAVE release of macOS, telnet is no longer included in the base OS and must be installed separately. You can download a telnet kit from: `http://osxdaily.com/2018/07/18/get-telnet-macros/`

To determine your host IP address, use the Apple network configuration utility: **Settings->Network**

The recommended connection sequence is:

1. Open your telnet session once you have reached the BOOTMGR> prompt on your Virtual Machine Guest. For example:

   $ telnet {your-host-ipv4-address} 2023

2. To enter telnet commands, you must be at the telnet> prompt. You can get to the prompt using an escape sequence, for example: **^]** (escape right bracket). On the Apple macOS, use the **Control** key to issue the escape character, therefore your command would be **Control]**

3. Turn off line-feeds using the command: **set crlf off**

4. Enter your escape sequence again to get another telnet> prompt

5. Set character mode using the command: **mode char**

Now from the BOOTMGR> prompt, enter the command: BOOTMGR> **COM 1** if you wish to use your terminal session with the Boot Manager. Once the boot begins, both OpenVMS and XDELTA will automatically use your telnet session.

CentOS Hosting VirtualBox Tip

When using CentOS as the Host for VirtualBox, we initially had trouble getting a serial port connection working.

In the VirtualBox Guest *Port* Setup, the COM 1 port is set up to forward serial traffic through TCP Port 2023, but it was not working.

It turned out that CentOS had not enabled port 2023. Below is the solution. This applies to CentOS Version 7, but probably works on other versions too.

You can see if port 2023 is enabled with the following command:

 # firewall-cmd --list-ports

```
23/tcp
```

As you can see, only port 23 was enabled.

If port 2023 was not listed, then the following two commands will do the trick. Note that the *--permanent* flag assures that the port will be enabled on subsequent boots.

 # firewall-cmd --permanent --add-port=2023/tcp

 # firewall-cmd --reload

Now check again with the *--list-ports* command to verify that it is now enabled.