Chapter 10. Technical Background

Table of Contents

This chapter provides additional information for readers who are familiar with computer architecture and technology and wish to find out more about how Oracle VM VirtualBox works *under the hood*. The contents of this chapter are not required reading in order to use Oracle VM VirtualBox successfully.

10.1. Where Oracle VM VirtualBox Stores its Files

In Oracle VM VirtualBox, a virtual machine and its settings are described in a virtual machine settings file in XML format. In addition, most virtual machines have one or more virtual hard disks. These are typically represented by disk images, such as those in VDI format. The location of these files may vary, depending on the host operating system. See Section 10.1.1, “The Machine Folder”.

Global configuration data for Oracle VM VirtualBox is maintained in another location on the host. See Section 10.1.2, “Global Settings”.

10.1.1. The Machine Folder

By default, each virtual machine has a directory on your host computer where all the files of that machine are stored: the XML settings file, with a `.vbox` file extension, and its disk images. This is called the *machine folder*.

By default, this machine folder is located in a common folder called `VirtualBox VMs`, which Oracle VM VirtualBox creates in the current system user's home directory. The location of this home directory depends on the conventions of the host operating system, as follows:

- On Windows, this is the location returned by the `SHGetFolderPath` function of the Windows system library Shell32.dll, asking for the user profile. A typical location is `C:\Users\`*`username`*.
- On Linux, macOS, and Oracle Solaris, this is generally taken from the environment variable `$HOME`, except for the user `root` where it is taken from the account database. This is a workaround for the frequent trouble caused by users using Oracle VM VirtualBox in combination with the tool **sudo**, which by default does not reset the environment variable `$HOME`.

  A typical location on Linux and Oracle Solaris is `/home/`*`username`* and on macOS is `/Users/`*`username`*.

For simplicity, we abbreviate the location of the home directory as `$HOME`. Using that convention, the common folder for all virtual machines is `$HOME/VirtualBox VMs`.

As an example, when you create a virtual machine called "Example VM", Oracle VM VirtualBox creates the following:

- A machine folder: `$HOME/VirtualBox VMs/Example VM/`

- In the machine folder, a settings file: `Example VM.vbox`
- In the machine folder, a virtual disk image: `Example VM.vdi`.

This is the default layout if you use the **Create New Virtual Machine** wizard described in Section 1.8, "Creating Your First Virtual Machine". Once you start working with the VM, additional files are added. Log files are in a subfolder called `Logs`, and if you have taken snapshots, they are in a `Snapshots` subfolder. For each VM, you can change the location of its snapshots folder in the VM settings.

You can change the default machine folder by selecting **Preferences** from the **File** menu in the Oracle VM VirtualBox main window. Then, in the displayed window, click on the **General** tab. Alternatively, use the **VBoxManage setproperty machinefolder** command. See Section 8.40, "VBoxManage setproperty".

10.1.2. Global Settings

In addition to the files for the virtual machines, Oracle VM VirtualBox maintains global configuration data in the following directory:

- **Linux and Oracle Solaris:** `$HOME/.config/VirtualBox`.
- **Windows:** `$HOME/.VirtualBox`.
- **macOS:** `$HOME/Library/VirtualBox`.

Oracle VM VirtualBox creates this configuration directory automatically, if necessary. You can specify an alternate configuration directory by either setting the `VBOX_USER_HOME` environment variable, or on Linux or Oracle Solaris by using the standard `XDG_CONFIG_HOME` variable. Since the global `VirtualBox.xml` settings file points to all other configuration files, this enables switching between several Oracle VM VirtualBox configurations.

In this configuration directory, Oracle VM VirtualBox stores its global settings file, an XML file called `VirtualBox.xml`. This file includes global configuration options and a list of registered virtual machines with pointers to their XML settings files.

10.1.3. Summary of Configuration Data Locations

The following table gives a brief overview of the configuration data locations on an Oracle VM VirtualBox host.

Table 10.1. Configuration File Locations

Setting	Location
Default machines folder	`$HOME/VirtualBox VMs`
Default disk image location	In each machine's folder
Machine settings file extension	`.vbox`
Media registry	Each machine settings file Media registration is done automatically when a storage medium is attached to a VM

10.1.4. Oracle VM VirtualBox XML Files

Oracle VM VirtualBox uses XML for both the machine settings files and the global configuration file, `VirtualBox.xml`.

All Oracle VM VirtualBox XML files are versioned. When a new settings file is created, for example because a new virtual machine is created, Oracle VM VirtualBox automatically uses the settings format of the current Oracle VM VirtualBox version. These files may not be readable if you downgrade to an earlier version of Oracle VM VirtualBox. However, when Oracle VM VirtualBox encounters a settings file from an earlier version, such as after upgrading Oracle VM VirtualBox, it attempts to preserve the settings format as much as possible. It will only silently upgrade the settings format if the current settings cannot be expressed in the old format, for example because you enabled a feature that was not present in an earlier version of Oracle VM VirtualBox.

In such cases, Oracle VM VirtualBox backs up the old settings file in the virtual machine's configuration directory. If you need to go back to the earlier version of Oracle VM VirtualBox, then you will need to manually copy these backup files back.

We intentionally do not document the specifications of the Oracle VM VirtualBox XML files, as we must reserve the right to modify them in the future. We therefore strongly suggest that you do not edit these files manually. Oracle VM VirtualBox provides complete access to its configuration data through its the **VBoxManage** command line tool, see Chapter 8, *VBoxManage* and its API, see Chapter 11, *Oracle VM VirtualBox Programming Interfaces*.

10.2. Oracle VM VirtualBox Executables and Components

Oracle VM VirtualBox was designed to be modular and flexible. When the Oracle VM VirtualBox graphical user interface (GUI) is opened and a VM is started, at least the following three processes are running:

- **VBoxSVC**, the Oracle VM VirtualBox service process which always runs in the background. This process is started automatically by the first Oracle VM VirtualBox client process and exits a short time after the last client exits. The first Oracle VM VirtualBox service can be VirtualBox Manager, **VBoxManage**, **VBoxHeadless**, the web service amongst others. The service is responsible for bookkeeping, maintaining the state of all VMs, and for providing communication between Oracle VM VirtualBox components. This communication is implemented using COM/XPCOM.

  > **Note**
  >
  > When we refer to *clients* here, we mean the local clients of a particular **VBoxSVC** server process, not clients in a network. Oracle VM VirtualBox employs its own client/server design to allow its processes to cooperate, but all these processes run under the same user account on the host operating system, and this is totally transparent to the user.

- The GUI process, **VirtualBoxVM**, a client application based on the cross-platform Qt library. When started without the `--startvm` option, this application acts as VirtualBox Manager, displaying the VMs and their settings. It then communicates settings and state changes to **VBoxSVC** and also reflects changes effected through other means, such as the **VBoxManage** command.

- If the **VirtualBoxVM** client application is started with the `--startvm` argument, it loads the VMM library which includes the actual hypervisor and then runs a virtual machine and provides the input and output for the guest.

Any Oracle VM VirtualBox front-end, or client, will communicate with the service process and can both control and reflect the current state. For example, either the VM selector or the VM window or VBoxManage can be used to pause the running VM, and other components will always reflect the changed state.

The Oracle VM VirtualBox GUI application, called VirtualBox Manager, is only one of several available front ends, or clients. The complete list shipped with Oracle VM VirtualBox is as follows:

- **VirtualBoxVM**: The Qt front end implementing VirtualBox Manager and running VMs.

- **VBoxManage**: A less user-friendly but more powerful alternative. See Chapter 8, *VBoxManage*.
- **VBoxHeadless**: A VM front end which does not directly provide any video output and keyboard or mouse input, but enables redirection through the VirtualBox Remote Desktop Extension. See Section 7.1.2, "VBoxHeadless, the Remote Desktop Server".
- **vboxwebsrv**: The Oracle VM VirtualBox web service process which enables control of an Oracle VM VirtualBox host remotely. This is described in detail in the Oracle VM VirtualBox Software Development Kit (SDK) reference. See Chapter 11, *Oracle VM VirtualBox Programming Interfaces*.
- The Oracle VM VirtualBox Python shell: A Python alternative to **VBoxManage**. This is also described in the SDK reference.

Internally, Oracle VM VirtualBox consists of many more or less separate components. You may encounter these when analyzing Oracle VM VirtualBox internal error messages or log files. These include the following:

- IPRT: A portable runtime library which abstracts file access, threading, and string manipulation. Whenever Oracle VM VirtualBox accesses host operating features, it does so through this library for cross-platform portability.
- VMM (Virtual Machine Monitor): The heart of the hypervisor.
- EM (Execution Manager): Controls execution of guest code.
- TRPM (Trap Manager): Intercepts and processes guest traps and exceptions.
- HM (Hardware Acceleration Manager): Provides support for VT-x and AMD-V.
- GIM (Guest Interface Manager): Provides support for various paravirtualization interfaces to the guest.
- PDM (Pluggable Device Manager): An abstract interface between the VMM and emulated devices which separates device implementations from VMM internals and makes it easy to add new emulated devices. Through PDM, third-party developers can add new virtual devices to Oracle VM VirtualBox without having to change Oracle VM VirtualBox itself.
- PGM (Page Manager): A component that controls guest paging.
- TM (Time Manager): Handles timers and all aspects of time inside guests.
- CFGM (Configuration Manager): Provides a tree structure which holds configuration settings for the VM and all emulated devices.
- SSM (Saved State Manager): Saves and loads VM state.
- VUSB (Virtual USB): A USB layer which separates emulated USB controllers from the controllers on the host and from USB devices. This component also enables remote USB.
- DBGF (Debug Facility): A built-in VM debugger.
- Oracle VM VirtualBox emulates a number of devices to provide the hardware environment that various guests need. Most of these are standard devices found in many PC compatible machines and widely supported by guest operating systems. For network and storage devices in particular, there are several options for the emulated devices to access the underlying hardware. These devices are managed by PDM.
- Guest Additions for various guest operating systems. This is code that is installed from within a virtual machine. See Chapter 4, *Guest Additions*.
- The "Main" component is special. It ties all the above bits together and is the only public API that Oracle

VM VirtualBox provides. All the client processes listed above use only this API and never access the hypervisor components directly. As a result, third-party applications that use the Oracle VM VirtualBox Main API can rely on the fact that it is always well-tested and that all capabilities of Oracle VM VirtualBox are fully exposed. It is this API that is described in the Oracle VM VirtualBox SDK. See Chapter 11, *Oracle VM VirtualBox Programming Interfaces*.

10.3. Hardware Virtualization

Oracle VM VirtualBox enables software in the virtual machine to run directly on the processor of the host, but an array of complex techniques is employed to intercept operations that would interfere with your host. Whenever the guest attempts to do something that could be harmful to your computer and its data, Oracle VM VirtualBox steps in and takes action. In particular, for lots of hardware that the guest believes to be accessing, Oracle VM VirtualBox simulates a certain *virtual* environment according to how you have configured a virtual machine. For example, when the guest attempts to access a hard disk, Oracle VM VirtualBox redirects these requests to whatever you have configured to be the virtual machine's virtual hard disk. This is normally an image file on your host.

Unfortunately, the x86 platform was never designed to be virtualized. Detecting situations in which Oracle VM VirtualBox needs to take control over the guest code that is executing, as described above, is difficult. To achieve this, Oracle VM VirtualBox uses *hardware virtualization*.

Intel and AMD processors have support for hardware virtualization. This means that these processors can help Oracle VM VirtualBox to intercept potentially dangerous operations that a guest operating system may be attempting and also makes it easier to present virtual hardware to a virtual machine.

These hardware features differ between Intel and AMD processors. Intel named its technology VT-x, AMD calls theirs AMD-V. The Intel and AMD support for virtualization is very different in detail, but not very different in principle.

> **Note**
>
> On many systems, the hardware virtualization features first need to be enabled in the BIOS before Oracle VM VirtualBox can use them.

Enabling hardware virtualization is *required* in the following scenarios:

- Certain rare guest operating systems like OS/2 make use of very esoteric processor instructions. For virtual machines that are configured to use such an operating system, hardware virtualization is enabled automatically.
- Oracle VM VirtualBox's 64-bit guest and multiprocessing (SMP) support both require hardware virtualization to be enabled. This is not much of a limitation since the vast majority of 64-bit and multicore CPUs ship with hardware virtualization. The exceptions to this rule are some legacy Intel and AMD CPUs.

> **Warning**
>
> Do not run other hypervisors, either open source or commercial virtualization products, together with Oracle VM VirtualBox. While several hypervisors can normally be *installed* in parallel, do not attempt to *run* several virtual machines from competing hypervisors at the same time. Oracle VM VirtualBox cannot track what another hypervisor is currently attempting to do on the same host, and especially if several products attempt to use hardware virtualization features such as VT-x, this can crash the entire host.

See Section 10.4, “Details About Hardware Virtualization” for a technical discussion of hardware virtualization.

10.4. Details About Hardware Virtualization

With Intel VT-x, there are two distinct modes of CPU operation: VMX root mode and non-root mode.

- In root mode, the CPU operates much like older generations of processors without VT-x support. There are four privilege levels, called rings, and the same instruction set is supported, with the addition of several virtualization specific instruction. Root mode is what a host operating system without virtualization uses, and it is also used by a hypervisor when virtualization is active.
- In non-root mode, CPU operation is significantly different. There are still four privilege rings and the same instruction set, but a new structure called VMCS (Virtual Machine Control Structure) now controls the CPU operation and determines how certain instructions behave. Non-root mode is where guest systems run.

Switching from root mode to non-root mode is called "VM entry", the switch back is "VM exit". The VMCS includes a guest and host state area which is saved/restored at VM entry and exit. Most importantly, the VMCS controls which guest operations will cause VM exits.

The VMCS provides fairly fine-grained control over what the guests can and cannot do. For example, a hypervisor can allow a guest to write certain bits in shadowed control registers, but not others. This enables efficient virtualization in cases where guests can be allowed to write control bits without disrupting the hypervisor, while preventing them from altering control bits over which the hypervisor needs to retain full control. The VMCS also provides control over interrupt delivery and exceptions.

Whenever an instruction or event causes a VM exit, the VMCS contains information about the exit reason, often with accompanying detail. For example, if a write to the CR0 register causes an exit, the offending instruction is recorded, along with the fact that a write access to a control register caused the exit, and information about source and destination register. Thus the hypervisor can efficiently handle the condition without needing advanced techniques such as CSAM and PATM described above.

VT-x inherently avoids several of the problems which software virtualization faces. The guest has its own completely separate address space not shared with the hypervisor, which eliminates potential clashes. Additionally, guest OS kernel code runs at privilege ring 0 in VMX non-root mode, obviating the problems by running ring 0 code at less privileged levels. For example the SYSENTER instruction can transition to ring 0 without causing problems. Naturally, even at ring 0 in VMX non-root mode, any I/O access by guest code still causes a VM exit, allowing for device emulation.

The biggest difference between VT-x and AMD-V is that AMD-V provides a more complete virtualization environment. VT-x requires the VMX non-root code to run with paging enabled, which precludes hardware virtualization of real-mode code and non-paged protected-mode software. This typically only includes firmware and OS loaders, but nevertheless complicates VT-x hypervisor implementation. AMD-V does not have this restriction.

Of course hardware virtualization is not perfect. Compared to software virtualization, the overhead of VM exits is relatively high. This causes problems for devices whose emulation requires high number of traps. One example is a VGA device in 16-color mode, where not only every I/O port access but also every access to the framebuffer memory must be trapped.

10.5. Paravirtualization Providers

Oracle VM VirtualBox enables the exposure of a paravirtualization interface, to facilitate accurate and efficient execution of software within a virtual machine. These interfaces require the guest operating system to recognize their presence and make use of them in order to leverage the benefits of communicating with the Oracle VM VirtualBox hypervisor.

Most modern, mainstream guest operating systems, including Windows and Linux, ship with support for one or

more paravirtualization interfaces. Hence, there is typically no need to install additional software in the guest to take advantage of this feature.

Exposing a paravirtualization provider to the guest operating system does not rely on the choice of host platforms. For example, the *Hyper-V* paravirtualization provider can be used for VMs to run on any host platform supported by Oracle VM VirtualBox and not just Windows.

Oracle VM VirtualBox provides the following interfaces:

- **Minimal**: Announces the presence of a virtualized environment. Additionally, reports the TSC and APIC frequency to the guest operating system. This provider is mandatory for running any Mac OS X guests.
- **KVM**: Presents a Linux KVM hypervisor interface which is recognized by Linux kernels version 2.6.25 or later. Oracle VM VirtualBox's implementation currently supports paravirtualized clocks and SMP spinlocks. This provider is recommended for Linux guests.
- **Hyper-V**: Presents a Microsoft Hyper-V hypervisor interface which is recognized by Windows 7 and newer operating systems. Oracle VM VirtualBox's implementation currently supports paravirtualized clocks, APIC frequency reporting, guest debugging, guest crash reporting and relaxed timer checks. This provider is recommended for Windows guests.

10.6. Nested Paging and VPIDs

In addition to normal hardware virtualization, your processor may also support the following additional sophisticated techniques:

- Nested paging implements some memory management in hardware, which can greatly accelerate hardware virtualization since these tasks no longer need to be performed by the virtualization software.

  With nested paging, the hardware provides another level of indirection when translating linear to physical addresses. Page tables function as before, but linear addresses are now translated to "guest physical" addresses first and not physical addresses directly. A new set of paging registers now exists under the traditional paging mechanism and translates from guest physical addresses to host physical addresses, which are used to access memory.

  Nested paging eliminates the overhead caused by VM exits and page table accesses. In essence, with nested page tables the guest can handle paging without intervention from the hypervisor. Nested paging thus significantly improves virtualization performance.

  On AMD processors, nested paging has been available starting with the Barcelona (K10) architecture. They now call it rapid virtualization indexing (RVI). Intel added support for nested paging, which they call extended page tables (EPT), with their Core i7 (Nehalem) processors.

  If nested paging is enabled, the Oracle VM VirtualBox hypervisor can also use *large pages* to reduce TLB usage and overhead. This can yield a performance improvement of up to 5%. To enable this feature for a VM, you use the **VBoxManage modifyvm --large-pages** command. See Section 8.10, “VBoxManage modifyvm”.

  If you have an Intel CPU with EPT, please consult Section 13.4.1, “CVE-2018-3646” for security concerns regarding EPT.

- On Intel CPUs, a hardware feature called Virtual Processor Identifiers (VPIDs) can greatly accelerate context switching by reducing the need for expensive flushing of the processor's Translation Lookaside Buffers (TLBs).

  To enable these features for a VM, you use the **VBoxManage modifyvm --vtx-vpid** and **VBoxManage**

modifyvm --large-pages commands. See Section 8.10, “VBoxManage modifyvm”.