

XenServer 5.5 Update 1 Driver Development Kit Guide

Overview

The XenServer Driver Development Kit (DDK) is used to develop or repackage existing modules for use with the XenServer line of virtualization platform products. XenServer is based on a standard Linux distribution, but for performance, maintainability, and compatibility reasons ad-hoc modifications to the core Linux components are not supported. As a result, operations that require recompiling the Linux kernel, such as adding device drivers, are not supported.

XenServer integrates the latest device support from kernel.org on a regular basis to provide a current set of device drivers. However, assuming appropriate redistribution agreements can be reached, there are situations where including additional device drivers in the shipping XenServer product, such as drivers not available through kernel.org, or drivers that have functionality not available through kernel.org, is greatly beneficial to joint customers of Citrix and the device manufacturer. The same benefits can apply by supplying device drivers independent of the XenServer product.

In addition, components such as command line interfaces (CLIs) for configuring and managing devices are also very valuable to include in the shipped XenServer product. Some of these components are simple binary RPM installs, but in many cases they are combined with the full driver installation making them difficult or impossible for administrators to install into XenServer. In either case including current versions of everything the administrator requires to use the device within the XenServer product provides significant value.

The DDK allows driver vendors to perform the necessary packaging and compilation steps with the XenServer kernel, which is not possible with the XenServer product alone. The resulting driver binary can then be installed into the XenServer product for verification testing, and the driver source and driver disk can be sent to the XenServer engineering team for simple inclusion in the shipping product.

Process

XenServer supports additional modules in the form of Red Hat Package Manager (RPM) packages. Whenever possible the RPMs should be source RPMs (SRPMs) so that they can be added to the standard XenServer build process and recompiled as changes are made to the XenServer kernel.

The high-level process of using the DDK to integrate a driver with XenServer is:

1. Obtain matching XenServer product and DDK build ISOs
2. Install the XenServer product onto a host server
3. Install the XenCenter administrator console onto a Windows machine
4. Use XenCenter to import the DDK into the XenServer host as a new virtual machine (VM)
5. Within the DDK VM, create a source RPM for the driver
6. Compile the source RPM into a binary RPM
7. Install the resulting binary RPM into the running XenServer host
8. Verify driver functionality within the XenServer host

If the driver is to be integrated with a pending XenServer release:

1. Supply the source RPM and any corresponding binary RPMs (such as proprietary CLIs) to the XenServer team

2. The XenServer team integrates the source and bin RPMs into the XenServer build and installation processes.
3. After receipt of a XenServer build with the RPMs integrated, perform any additional required verification testing.

Or, if the driver is to be supplied on a driver disk independent of the XenServer distribution media

1. Create a driver disk
2. Test the use of the driver disk during XenServer installation

Using the DDK

Installation

Installing the XenServer host

Installing the XenServer host only requires booting from the provided CD-ROM image and answering a few basic questions. After setup completes take note of the host IP address shown as it is required for connection from the Admin Console.

Intel VT or AMD-V CPU support is required to run Windows guests, but is not required in order to use the DDK and test drivers in the XenServer control domain (dom0).

Installing the XenCenter administrator console

XenCenter, the XenServer administration console, must be installed on a separate Windows machine. Inserting the XenServer installation CD will run the XenCenter installation automatically. The XenCenter console will be displayed with no servers connected.

Connect XenCenter to the XenServer host

Within XenCenter select the **Server > Add** menu option and supply the appropriate host name/IP address and password for the XenServer host.

Select the newly connected host in the left-hand tree view.

Importing the DDK VM through XenCenter

Note

You can also import the DDK directly on the host using the **xe** Command Line Interface (CLI).

- Insert the DDK CD into the CD-ROM drive of the machine running XenCenter.
- Select the **VM > Import** menu option. The VM Import Wizard is displayed:

- Click **Browse** and change the **Files of type** dropdown from the default of **XenServer Virtual Appliance (*.xva)** to **XenServer Virtual Appliance Version 1 (ova.xml)**.
- Navigate to the DDK CD-ROM and select the ova.xml file within the DDK directory.
- Click **Next** to use the defaults on the Home Server and Storage pages.
- On the Network page, add a virtual network interface to the VM.
- Finish the VM Import Wizard.

The DDK VM will be started automatically.

Using the DDK VM

After the import process completes the DDK VM will be started automatically. Select the VM in the left pane and then select the Console tab in the right pane to display the DDK VM's console to provide a terminal window in which you can work.

The DDK VM is Linux-based so you are free to use other methods such as ssh to access the DDK VM. You can also access the DDK VM console directly from the host console.

Importing the DDK VM using the CLI

The DDK VM can also be imported directly on the XenServer host using the xe CLI and standard Linux commands to mount the DDK ISO.

- Mount the DDK ISO and import the DDK VM:

```
mkdir /mnt/tmp  
mount <path to DDK ISO>/ddk.iso /mnt/tmp -o loop  
xe vm-import filename=/mnt/tmp/ddk/ova.xml
```

The uuid of the DDK VM is returned when the import completes.

- Add a virtual network interface to the DDK VM:

```
xe network-list
```

Note the uuid of the appropriate network to use with the DDK VM, typically this will be the network with a name-label of Pool-wide network associated with eth0.

```
xe vif-create network-uuid=<network_uuid> vm-uuid=<ddk_vm_uuid> device=0
```

Note

Use tab completion to avoid entering more than the first couple characters of the uuids.

Accessing the DDK VM console from the host console

The DDK VM text console can be accessed directly from the XenServer host console instead of using XenCenter. Note that using this method disables access to the DDK console from XenCenter.

- While the DDK VM is shutdown disable VNC access on the VM record:

```
xe vm-param-set uuid=<ddk_vm_uuid> other-config:disable_pv_vnc=1
```

- Start the VM

```
xe vm-start uuid=<ddk_vm_uuid>
```

- Retrieve the underlying domain ID of the VM:

```
xe vm-list params=dom-id uuid=<ddk_vm_uuid> --minimal
```

- Connect to the VM text console:

```
/usr/lib/xen/bin/xenconsole <dom-id>
```

- When complete, exit the xenconsole session using **CTRL-]**

For more information on using with the xe CLI please see the *XenServer Administrator's Guide* available online at <http://docs.xensource.com/XenServer/5.0.0/>

Creating a source RPM

Required variables

The RPM spec file should expect to be passed two variables (via the **rpmbuild --define** command line option):

Variable	Description
kernel_version	The version of the kernel RPM to build against
Flavour	The flavor of kernel to build against

For example, the `kernel-xen-2.6.16.33-xs0.4.0.316.3686.i686.rpm` RPM gives a `kernel_version` of `2.6.16.33-xs0.4.0.316.3686`.

The `kernel-xen-2.6.16.33-xs0.4.0.316.3686.i686.rpm` RPM gives a flavor of `xen`. Other flavors might be `xen0` or `kdump`.

The uname of the kernel to build against is therefore `%{kernel_version}%{?flavor}`.

For example the uname of the kernel in the `kernel-xen-2.6.16.33-xs0.4.0.316.3686.i686.rpm` RPM is `2.6.16.33-xs0.4.0.316.3686xen`.

Package naming

The binary package name should be suffixed with the flavor (if flavor is defined) and the `kernel_version`:

```
F00-modules-%{flavour}-%{kernel_version}
```

If flavor is not defined it should be omitted:

```
F00-modules-%{kernel_version}
```

Declarations

The binary package should declare that it provides:

```
F00-modules-%{flavour} = %{kernel_version}
```

and that it requires:

```
kernel-%{flavour} = %{kernel_version}
```

If no flavor is given this becomes provides:

```
foo-modules = ${kernel_version}
```

and requires:

```
kernel = %{kernel_version}
```

Building the SRPM

The .SPEC file should expect to find the kernel headers to build against in the standard upstream location of `/lib/modules/%{uname}/build`. The package should not attempt to build against the currently running kernel.

It should be possible to take the `.src.rpm` file and run the command:

```
rpmbuild --define "kernel_version 2.6.16.33-xs0.4.0.316.3686" \  
--define "flavour xen" \  
--rebuild F00.src.rpm
```

Sample .SPEC file

A sample RPM .SPEC file for modules that use the standard Linux 2.6 external module Makefile structure is below.

The same sample is also available in the `/root/README.ExternalModules` file within the DDK VM.

```
%define binsuffix %{?flavour:-%{flavour}}
%define uname    %{kernel_version}%{?flavour}

Summary: Driver for F00
Name: F00
Version: 1.0
Release: 1
License: GPL
Group: System Environment/Kernel
Source: %{name}-%{version}.tar.gz
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-buildroot

%description
F00 Linux Device Driver source.

%package modules%{binsuffix}-%{kernel_version}
Summary: The foo modules
Group: System Environment/Kernel
Provides: %{name}-modules%{binsuffix} = %{kernel_version}
Requires: kernel%{binsuffix} = %{kernel_version}

%description modules%{binsuffix}-%{kernel_version}
F00 Linux Device Driver modules compiled against kernel version %{uname}.

%prep
%setup -q -n %{name}-%{version}

%build
%{__make} -C /lib/modules/%{uname}/build M=$(pwd) modules

%install
rm -rf $RPM_BUILD_ROOT
%{__make} -C /lib/modules/%{uname}/build M=$(pwd) \
INSTALL_MOD_PATH=$RPM_BUILD_ROOT modules_install

# mark modules executable so that strip-to-file can strip them
find $RPM_BUILD_ROOT/lib/modules/%{uname} -name "*.ko" -type f | xargs chmod u+x

%clean
rm -rf $RPM_BUILD_ROOT

%post modules%{binsuffix}-%{kernel_version}
depmod %{uname}

%files modules%{binsuffix}-%{kernel_version}
%defattr(-,root,root,-)
/lib/modules/%{uname}/extra/*.ko

%doc

%changelog
```

Creating a driver disk

To support the use of drivers that are made available outside of the XenServer distribution media, XenServer setup allows the use of 3rd party driver disks. If the driver is to be integrated with a pending XenServer release this step is not necessary.

Please see the `/root/README.RepositoryFormat` file within the DDK VM for details on how to create a driver disk and the example driver disk found at `/root/examples`. Use `/root/examples/Makefile` to build the example driver disk. The makefile targets are:

`build-srctarballs`

Builds driver source tarballs.

`build-rpms`

Builds a RPM for installing the driver.

`build-iso`

Builds a driver disk iso image for distribution.

The driver disk example also generates `install.sh`, which can be used to install drivers on a running XenServer.

Testing

Overview

XenServer uses a modified Linux kernel that is similar but not identical to the kernel distributed by the distro that the XenServer control domain is based on. In addition, the XenServer control domain kernel is running above the 80K lines of code that are the Xen hypervisor itself. While we are very confident in the stability of the hypervisor, its presence represents a different software installation than exists with the stock vendor kernel installed on bare hardware.

In particular, there are issues that may be taken for granted on an x86 processor, such as the difference between physical and device bus memory addresses (e.g. `virt_to_phys()` vs. `virt_to_bus()`), timing, and interrupt delivery which may have subtle differences in a hypervisor environment.

For this reason hardware drivers should be exposed to a set of testing on the XenServer control domain kernel to ensure the same level of confidence that exists for drivers in enterprise distro releases.

Scope

Assuming the driver in question has already undergone verification testing on a distro very similar to the one used in the XenServer control domain, a subset of the verification test suite with a focus on representative tests is typically sufficient.

Some common areas of focus are:

- *Installation verification.* Installation is now performed via an RPM, which may be different from how the driver is typically installed.
- *CLI operation.* If a CLI is included its operation is a key scenario and typically provides a good end-to-end exercising of all related code.
- *Adapter configuration, non-volatile / flash RAM, or BIOS upgrades.* Any functions that access the physical hardware should be verified due to the presence of the Xen hypervisor and its role in coordination of hardware resources amongst virtual machines.
- *Data integrity and fault injection.* Long-haul and/or stress-based data integrity verification tests to verify no data corruption issues exist. Basic fault injection tests such as cable un-plug/re-plug and timeout handling for verification of common error conditions.

- *Coverage of a representative set of device models.* For example, if a Fibre Channel HBA driver supports a set of models that operate at either 2Gb or 4Gb, include one model each from the 2Gb and 4Gb families for testing.
- *Key hardware configuration variations.* Any hardware configurations that exercise the driver or related code in significantly different ways should be run, such as locally vs. remotely attached storage.

Process

Building the binary RPM

After the SRPM is completed it can be used to compile a binary RPM, which can then be installed in the XenServer host. The compilation can be done within the DDK by the team that created the SRPM, the team doing the verification testing, or both.

Important

The XenServer build into which the binary RPM is installed *must* be the same build as the DDK used to compile the SRPM.

Installing the binary RPM into the XenServer control domain

After the binary RPM is created, select the XenServer host in the left pane of the XenCenter and then click on the **Console** tab in the right pane. This provides you access to the XenServer control domain console. From there you can use standard methods to access and install the binary RPM. This is the functional equivalent of what will happen when the SRPM is integrated with the XenServer build process.

Running tests

Since the physical device drivers run in the XenServer control domain, the majority of tests will also be run in the control domain. This allows simple re-use of existing Linux-based tests.

To provide high-performance device I/O to guest domains, XenServer includes synthetic device drivers for storage and networking that run in a guest and communicate their I/O requests with corresponding back-end drivers running in the control domain. The back-end drivers then issue the I/O requests to the physical drivers and devices and manage transmitting results and completion notifications to the synthetic drivers. This approach provides near bare-metal performance.

As a result, tests that require direct access to the device will fail when run within a guest domain. However, running load-generation and other tests that do not require direct access to the device and/or driver within Linux and Windows guest domains is very valuable as such tests represent how the majority of load will be processed in actual XenServer installations.

When running tests in guest domains, ensure that you do so with the XenServer synthetic drivers installed in the guest domain. Installation of the synthetic drivers is a manual process for some guests. See XenServer Help for more details.

Tests that require an integrated build

One of the primary goals of the DDK is to allow partners to create, compile, and test their drivers with XenServer without requiring a “back-and-forth” of components with the XenServer engineering team.

However, some tests will only be possible after the driver SRPM and any accompanying binary RPMs have been supplied to Citrix and integrated into the XenServer product. Two examples are installing to and booting from Fibre Channel and iSCSI LUNs.

In these cases additional coordination is required after the components have been provided to Citrix to provide a pre-release XenServer build with the integrated components for testing.