

# Xen Cloud Platform Driver Development Kit Guide

## Overview

The Xen Cloud Platform Driver Development Kit (DDK) is used to develop or repackage existing modules for use with the Xen Cloud Platform line of virtualization platform products. Xen Cloud Platform 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.

Xen Cloud Platform 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 Xen Cloud Platform 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 Xen Cloud Platform product.

In addition, components such as command line interfaces (CLIs) for configuring and managing devices are also very valuable to include in the shipped Xen Cloud Platform 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 Xen Cloud Platform. In either case including current versions of everything the administrator requires to use the device within the Xen Cloud Platform product provides significant value.

The DDK allows driver vendors to perform the necessary packaging and compilation steps with the Xen Cloud Platform kernel, which is not possible with the Xen Cloud Platform product alone. This in turn enables the end user to install the resulting driver.

## Process

Xen Cloud Platform 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 Xen Cloud Platform build process and recompiled as changes are made to the Xen Cloud Platform kernel.

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

1. Obtain matching Xen Cloud Platform product and DDK build ISOs
2. Install the Xen Cloud Platform product onto a host server
3. Import the DDK into the Xen Cloud Platform host as a new virtual machine (VM)
4. Within the DDK VM, create a source RPM for the driver
5. Build a driver disk

6. Install the resulting driver disk onto a test Xen Cloud Platform host
7. Verify driver functionality within the Xen Cloud Platform host

If the driver is to be integrated with a pending Xen Cloud Platform release:

1. Supply the source RPM and any corresponding binary RPMs (such as proprietary CLIs) to the Xen Cloud Platform team
2. The Xen Cloud Platform team integrates the source and binary RPMs into the Xen Cloud Platform build and installation processes.
3. After receipt of a Xen Cloud Platform build with the RPMs integrated, perform any additional required verification testing.

## Using the DDK

### Installation

#### Installing the Xen Cloud Platform host

Installing the Xen Cloud Platform 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 Xen Cloud Platform control domain (`dom0`).

#### 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 be imported directly on the Xen Cloud Platform 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 Xen Cloud Platform host console. 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 *Xen Cloud Platform Administrator's Guide*.

## Creating a driver disk

An example driver disk is provided in the DDK at `/root/example`. Xen.org recommends using this example to familiarize yourself with the process of creating a driver disk before attempting to create your own.

The initial requirements for a driver disk are:

- The source code of the kernel module (`/root/example/helloworld-1.0` in the example).
- A specification file that describes how to build the source of the kernel RPM (`/root/example/helloworld.spec` in the example).

Other files can be included in a data RPM. The data RPM in the driver disk example includes:

- The source code of the data file (`/example/driver/helloworld-data-1.0`).
- A specification file that describes how to build the data RPM (`/root/example/helloworld-data.spec` in the example).

## Driver disk construction

The process of building and packaging a driver disk is controlled by the makefile in the example directory. Certain variables must be customized for your driver disk:

### SPEC

The kernel module specification filename

### DATASPEC

the data specification filename

### VENDOR

the name of the driver disk publisher

### LABEL

the identifier of the driver disk. By default this is the same as the name of the kernel module.

### TEXT

Free-form descriptive text which is displayed when the driver is installed.

### RPM.VERSION / RPM.RELEASE

Kernel RPM version information. Versioning is described in [the section called “Versioning”](#).

### DATA.RPM.VERSION / DATA.RPM.RELEASE

Data RPM version. Versioning is described in [the section called “Versioning”](#).

## Building the RPMs

Each kernel module RPM must be built against multiple kernel versions. The example driver disk provides a makefile target `build-rpms` which automates the build. The data RPM is also built if necessary.

### Creating the kernel module RPM

The kernel module packages are built according to the instructions in the specification file. The following sections of the specification file affect the building of a package and should be set to appropriate values:

- `Name` is a unique ID, preferably the name of the kernel module.
- `Summary` is a short description of the driver
- `%files` is a list of files that are to be compiled into the RPM. Kernel modules must be located in a directory called `extra`
- `%changelog` describes changes that have been made to the driver

### Creating the data RPM

Use the data RPM to install the following types of files that might need to accompany a device driver:

- udev rules must be located in `/etc/udev/rules.d`
- firmware must be located in `/lib/firmware`
- configuration must be located in `/etc`
- documentation must be located in `/usr/share/doc`

Multiple versions of a kernel module RPM can be installed but only a single instance of the data RPM can be installed for any given driver.

### Preparing source for building

Copy the specification files and sources for the kernel module and data RPMS to the appropriate sub-directory of `/usr/src/redhat`. The example driver disk uses the `build-srctarballs` target to do this.

### Generating the driver disk

Driver disks use a format described in the `/root/README.Repositoryformat` file. The example driver disk uses the `build-isos` target to generate a driver disk that combines the source and data RPMs. This target generates the following files:

- `<driver_name>.iso` is the driver disk ISO that can be installed using the Xen Cloud Platform installer
- `<driver_name>.md5` is a checksum that users can use to validate the driver disk

## Testing

### Overview

Xen Cloud Platform uses a modified Linux kernel that is similar but not identical to the kernel distributed by the distro that the Xen Cloud Platform control domain is based on. In addition, the Xen Cloud Platform 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 (for example `virt_to_phys()` as opposed to `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 Xen Cloud Platform 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 Xen Cloud Platform 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 using 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 unplug/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 versus remotely attached storage.

## Process

### Building the driver disk

Build your driver disk as described in this document. This can be done within the DDK by the team that created the SRPM, the team doing the verification testing, or both.

---

#### **Important**

The Xen Cloud Platform build into which the driver disk is installed *must* be the same build as the DDK used to build it.

---

### Installing the driver disk

Test driver disk installation both using the Xen Cloud Platform installer and onto a running Xen Cloud Platform host.

### Using the Xen Cloud Platform installer

The Xen Cloud Platform installer can load driver disks from either local media (CD, USB) or from a remote system over FTP, HTTP or NFS. The ISO created by the process above can be burned on to a CD. Alternately the files it contains can be copied onto a USB device.

To access a driver disk in the network the contents of the CD should be copied to an accessible directory.

## Onto a running Xen Cloud Platform host

A driver disk contains an installation script called `install.sh`. Copy the contents of the driver disk on to your host and run `install.sh` to install the driver.

## Running tests

Since the physical device drivers run in the Xen Cloud Platform 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, Xen Cloud Platform 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 Xen Cloud Platform installations.

When running tests in guest domains, ensure that you do so with the Xen Cloud Platform synthetic drivers installed in the guest domain. Installation of the synthetic drivers is a manual process for some guests. See Xen Cloud Platform 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 Xen Cloud Platform without requiring a “back-and-forth” of components with the Xen Cloud Platform engineering team.

However, some tests will only be possible after the driver SRPM and any accompanying binary RPMs have been supplied to Xen.org and integrated into the Xen Cloud Platform 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 Xen.org to provide a pre-release Xen Cloud Platform build with the integrated components for testing.

## Versioning

### Driver disks

Driver disks are built for a specific Xen Cloud Platform release and may only be installed on that version. Hotfixes do not change the version number, so you can construct a driver disk that supports a GA release and all subsequent hotfixes.

## Kernel modules

Kernel modules must be built for two kernels, the Xen kernel that a Xen Cloud Platform host usually runs and the **kdump** kernel used to collect a crash dump. The makefile in the example ensures that RPMs for both flavours are built.

Each kernel module is built against a specific kernel version. This kernel version is included in the RPM name to enable multiple instances to be installed. The version fields of a kernel module RPM are used to track changes to a driver for a single kernel version. Each time a driver is released for a particular kernel the RPM version must be increased. When a Xen Cloud Platform release updates the kernel the RPM version should revert to 1.0.

For example:

```
helloworld-modules-xen-2.6.18-128.1.6.el5.xs5.5.0.502.1014-1.0-1.i386.rpm
driver name = helloworld
kernel flavour = xen
kernel version = 2.6.18-128.1.6.el5.xs5.5.0.502.1014
RPM version = 1.0-1
```

Kernel version	RPM version	Event
2.6.18.128.1.5...	1.0-1	Initial release
2.6.18.128.1.5...	2.0-1	Driver bug fix
2.6.18.128.1.6...	1.0-1	New kernel

## Data RPMS

Only a single instance of a data RPM can be installed on a Xen Cloud Platform host therefore the RPM is versioned as usual.

```
helloworld-data-1.0-1.i386.rpm
driver name = helloworld
RPM version = 1.0-1
```