Disclaimer and Legal Information

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel, the Intel logo, Intel Xeon, Xeon Phi and vTune are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.
# Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Description</th>
<th>Revision Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>Document update for the release of the Intel® MPSS 3.8 for Windows*</td>
<td>July 2016</td>
</tr>
<tr>
<td>3.7</td>
<td>Initial draft of the 3.7 Intel® MPSS User’s Guide for Windows*</td>
<td>January 2016</td>
</tr>
<tr>
<td>3.6</td>
<td>Initial draft of the 3.6 Intel® MPSS User’s Guide</td>
<td>August 2015</td>
</tr>
</tbody>
</table>
## Table of Contents

1  Introduction .................................................................................................................. 8  
   1.1  Conventions and symbols ...................................................................................... 8  
2  Post installation quick configuration ........................................................................... 10  
   2.1  Step 1: ensure admin access ................................................................................. 10  
   2.2  Step 2: change configuration ................................................................................ 10  
   2.3  Step 3: start the coprocessors .............................................................................. 10  
3  Intel® MPSS boot process ............................................................................................. 11  
   3.1  Booting the coprocessor ....................................................................................... 11  
      3.1.1  Kernel command line ...................................................................................... 12  
      3.1.2  Instruct the driver to boot the coprocessor .................................................... 12  
      3.1.3  Linux* kernel executables ............................................................................ 12  
      3.1.4  Root is the initial RAM disk ......................................................................... 13  
      3.1.5  Root is a RAM disk image ............................................................................ 13  
      3.1.6  Notify the host that the coprocessor is ready ................................................ 13  
4  Configuration ............................................................................................................... 14  
   4.1  Configurable components ...................................................................................... 14  
   4.2  Configuration files ................................................................................................ 14  
      4.2.1  File location and format ............................................................................... 14  
   4.3  Configuring boot parameters ................................................................................ 14  
      4.3.1  Power management ....................................................................................... 15  
      4.3.2  Command line ............................................................................................... 15  
      4.3.3  Networking ................................................................................................... 15  
   4.4  Root file system .................................................................................................... 15  
   4.5  Routing .................................................................................................................. 15  
   4.6  Bridging ................................................................................................................. 16  
      4.6.1  Internal bridging ............................................................................................. 16  
      4.6.2  External bridging ............................................................................................ 17  
5  The micctrl utility .......................................................................................................... 18  
   5.1  Coprocessor state control ...................................................................................... 18  
      5.1.1  Booting the coprocessor ............................................................................... 18  
      5.1.2  Resetting the coprocessor ............................................................................. 18  
      5.1.3  Waiting for the coprocessor’s state change ...................................................... 19  
      5.1.4  Checking coprocessor status ........................................................................ 19  
   5.2  Helper functions for configuration parameters ...................................................... 19  
      5.2.1  Adding users to the coprocessor’s file system ................................................ 19  
      5.2.2  Removing users from the coprocessor’s file system ...................................... 19  
      5.2.3  Specifying the host secure shell keys ............................................................ 19  
   5.3  Other file system helper functions ....................................................................... 20  
      5.3.1  Updating the compressed cpio image ........................................................... 20  
6  Adding software ........................................................................................................... 21  
   6.1  The file system creation process .......................................................................... 21  
      6.1.1  The dir filelist directive ................................................................................ 21  
      6.1.2  The file filelist directive ............................................................................... 22  
      6.1.3  The slink filelist directive ............................................................................. 22  

Intel® Manycore Platform Software Stack (Intel® MPSS)  
User’s Guide (Windows*)  
March 2017  
Document Number: 330077-001US
6.1.4 The nod filelist directive ................................................................. 22
6.1.5 The pipe filelist directive .............................................................. 23
6.1.6 The sock filelist directive .............................................................. 23
6.2 Adding persistent files to the coprocessor’s file system ....................... 23
6.3 Creating the download image file ...................................................... 24
6.4 Adding files to the root file system .................................................... 24
6.4.1 Adding files by copying ............................................................... 24

7 Intel® Xeon Phi™ coprocessor tools and utilities .................................... 25
7.1 Coprocessor information tool: micinfo ............................................. 25
7.1.1 Simple method ......................................................................... 25
7.1.2 For advanced users .................................................................. 25
7.2 The coprocessor platform status panel: micsmc ................................. 26
7.3 The coprocessor verification tool: miccheck ...................................... 26
7.4 The coprocessor RAS tool: micras .................................................. 26
7.4.1 Simple method ......................................................................... 26
7.4.2 For advanced users .................................................................. 26
7.5 The coprocessor utility: micnativeloadex ......................................... 28

8 Host file system share ........................................................................ 30
8.1 Creating an NFS share .................................................................... 30
8.1.1 Adding the NFS server .............................................................. 30
8.1.2 Provisioning a folder for the NFS share ....................................... 33
8.1.3 Specifying the NFS share permissions ....................................... 36
8.1.4 Using Powershell* cmdlets to create an NFS share (optional) ....... 38
8.2 Mounting the NFS share ............................................................... 39

9 CIFS share on windows* ..................................................................... 40
9.1 Mounting CIFS shares with micctrl ................................................ 40
9.1.1 Mounting a temporary CIFS share ............................................. 43
9.1.2 Troubleshooting ....................................................................... 43
9.2 Configuring a CIFS share with a bridge .......................................... 44
9.3 Unmounting a CIFS share with micctrl .......................................... 44

10 Interacting with the Intel® Xeon Phi™ coprocessor .............................. 45
10.1 External tools ................................................................................ 45
10.2 Special user account micuser ........................................................... 45
10.3 SSH access to the coprocessor ........................................................ 45
10.4 Transferring files to and from the coprocessor ................................ 48
10.5 User groups .................................................................................. 50
10.5.1 Deleting users from the micusers group .................................... 50
10.6 Running the COI tutorials (optional) ............................................... 51
10.7 Running the MYO tutorials (optional) ............................................. 51
10.8 Running the SCIF tutorials (optional) ............................................. 51
10.9 Running the MicMgmt tutorials (optional) ....................................... 52
10.10 Troubleshooting ......................................................................... 52

11 Installing Intel® Xeon Phi™ coprocessor Performance Workloads (optional) 54
11.1 Requirements ............................................................................... 54
11.2 Distributed files ........................................................................... 55
11.3 micp python installation ............................................................... 55
12 **Important considerations** ......................................................................................... 57
12.1 Disabling and enabling the Memory Control Group (cgroup) ............................... 57
12.2 Enabling Windows® MIC GDB debugging for offload processes ...................... 58
12.3 Enabling Windows® MIC debugging for MYO applications ............................... 58
12.4 Copying RPMs to the coprocessor using pscp ....................................................... 59

13 **Related documentation** ....................................................................................... 60
13.1 MYO documentation ............................................................................................... 60
13.2 COI documentation ............................................................................................... 60
13.3 Intel® Xeon Phi™ coprocessor collateral ............................................................... 60
List of Tables

Table 1 Conventions and symbols ................................................................. 8
Table 2 micras severity levels ..................................................................... 28

List of Figures

Figure 1 Coprocessor boot process ............................................................. 11
Figure 2 Internal bridge network ................................................................. 16
Figure 3 External bridge network ................................................................. 17
1 Introduction

The Intel® Xeon Phi™ coprocessors are PCIe based add-in cards that run a version of Linux* tailored for specifically for them. The operating system for the coprocessors, as well as a range of drivers and utilities, are included in the Intel® Manycore Platform Software Stack (Intel® MPSS). The responsibilities of these drivers and utilities include:

- Placing the Linux* boot image and root file system into coprocessor memory.
- Controlling coprocessor boot, shutdown and reset.
- Providing an IP (over PCIe) networking connection to each coprocessor.
- Directing power management of each coprocessor.
- Supporting high speed data transfer to and from the coprocessor.

The PCIe bus is the only communication channel available to the coprocessors. Therefore configuration and provisioning of the OS to be executed on each coprocessor is performed by the host system in which it is installed.

The Linux* kernel and file system image for the coprocessors are installed into the host file system as part of Intel® MPSS installation. The coprocessor file system image can be configured through the use of the micctrl utility described in this paper and/or directly by the host root.

The micx64.sys driver is the component of the software stack that provides PCIe bus access and implements the coprocessor boot process. To boot a coprocessor, micx64.sys injects the Linux* kernel image and a kernel command line into its memory and signals it to begin execution. A virtual network driver is installed as micvethx64.sys. Finally, micx64.sys directs power management of the installed coprocessors and provides a high speed data transfer over PCIe through its Intel® Symmetric Communications Interface (SCIF) driver.

micctrl is a utility through which the user can control (boot, shutdown, reset) each of the installed coprocessors. micctrl also offers numerous options to simplify the process of configuring each coprocessor. Section 5 of this document describes the micctrl utility in detail.

1.1 Conventions and symbols

Table below lists conventions used in this document.

<table>
<thead>
<tr>
<th>Table 1 Conventions and symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>This type style</td>
</tr>
<tr>
<td>This type style</td>
</tr>
</tbody>
</table>
**This type style**

Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder. Also used to indicate new terms, URLs, email addresses, filenames, and file extensions.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ items ]</td>
<td>Indicates that the items enclosed in brackets are optional.</td>
</tr>
<tr>
<td>{ item</td>
<td>item }</td>
</tr>
<tr>
<td>... (ellipses)</td>
<td>Indicates that you can repeat the preceding item.</td>
</tr>
<tr>
<td>\ (backslash)</td>
<td>Indicates continuation of a command onto the next line in the document.</td>
</tr>
<tr>
<td>micN</td>
<td>Denotes a name such as mic0, mic1, etc. where N=0,1, 2, ... For example: the file name micN.conf denotes file names mic0.conf, mic1.conf, etc.</td>
</tr>
<tr>
<td>User&gt;</td>
<td>Denotes a command entered on the host with user privileges.</td>
</tr>
<tr>
<td>Admin&gt;</td>
<td>Denotes a command entered on the host with administrative (root) privileges.</td>
</tr>
<tr>
<td>[micN]$</td>
<td>Denotes a command entered on a coprocessor N with user privileges.</td>
</tr>
<tr>
<td>[micN]#</td>
<td>Denotes a command entered on a coprocessor N with administrative privileges.</td>
</tr>
<tr>
<td>*</td>
<td>Other names and brands may be claimed as the property of others.</td>
</tr>
</tbody>
</table>

* Other names and brands may be claimed as the property of others.
2 Post installation quick configuration

After the installation of the software stack (consult the readme-windows.pdf file for installation instructions), the system administrator must complete the configuration before starting the coprocessors.

2.1 Step 1: ensure admin access

Users must be added to the MICUSERS group in order to log into the coprocessor (refer to Section 10.5 for instructions on how to create and modify the MICUSERS group).

User access to the coprocessor node is provided through the secure shell utilities. Ensure the admin user has SSH keys. If no such keys exist, generate a set with an external utility (refer to Section 10.3 for instructions on generating SSH keys using PuTTY).

2.2 Step 2: change configuration

Examine the global.xml and micN.xml files in the C:\Program Files\Intel\MPSS directory. If during installation the base install directory was changed from the default, this path will differ. If the default configuration meets the requirements of the system, continue to Section 2.3. Otherwise, edit the configuration files (refer to Section 4). If the micN.xml files do not exist, generate them by executing the following command as an Administrator:

Admin> micctrl -g

2.3 Step 3: start the coprocessors

The default configuration specifies that each coprocessor is booted when the host driver is loaded. This means that the coprocessors will boot when the host system restarts. To start the coprocessors manually, execute the following command as an Administrator:

Admin> micctrl --start

The call to micctrl will exit when it determines the coprocessors have either booted successfully or failed to boot.
3 Intel® MPSS boot process

Booting the Linux* kernel on the Intel® Xeon Phi™ coprocessor requires a number of steps. Figure 1 shows the sequence of steps that are performed during the boot process.

![Diagram of the Intel® MPSS boot process]

Figure 1 Coprocessor boot process

3.1 Booting the coprocessor

This section describes the key steps that are performed during the Intel® MPSS boot process on the coprocessor.
3.1.1 Kernel command line

On most Linux* based systems, loading and executing the kernel image is controlled by the grub boot loader. In the grub configuration file, each possible kernel definition contains a number of parameters to be passed to Linux* through its kernel command line. In the Intel® MPSS boot system, this is done by parsing the configuration files. The kernel command line is created based on values in the configuration files and placed in the WMI entry `cmdline` for the driver to retrieve it.

3.1.2 Instruct the driver to boot the coprocessor

The `micctrl` utility requests the coprocessor to start executing the Linux* image by executing the BootMIC WMI method. This method is a link into the MIC driver through WMI.

The method ResetMIC may also be executed and will be discussed later.

When the driver receives the boot request, it first checks to see whether the coprocessor is in the `ready` state. If the coprocessor is not ready to boot, it will return an error from WMI and will not attempt to perform the boot. Otherwise the state of the coprocessor is set to `booting`.

Next the driver saves the image file name for later retrieval through the image WMI entry. It also sets the mode to indicate it is booting a Linux* image.

The driver will copy the kernel command line setting request from the `micctrl` utility, along with a number of addresses in host memory required by various drivers in the Linux* image. It then copies the requested Linux* image file into the coprocessor’s memory.

The last step is to write to the coprocessor’s register instructing it to start executing the injected image.

3.1.3 Linux* kernel executables

Executing the Linux* kernel code functions is performed as on any Intel® based machine. It initializes hardware, starts kernel services, and sets each coprocessor to the `online` state. When the kernel is ready, it initializes its attached initial ram disk image and starts executing the `init` script in the image.

The initial ram disk contains the loadable modules required for the real root file system. Many of the arguments passed in the kernel command line are addresses required for the modules to access host memory. The `init` script parses the kernel command line for needed information and loads the driver modules.

The last step is for the `init` script to check the `root=` parameter in the kernel command line for the type of device containing the root file system, and take the appropriate actions.
3.1.4 Root is the initial RAM disk

Setting the root to be the initial ram disk is for debug purposes only. The initial ram disk contains only a minimal set of tools and utilities.

3.1.5 Root is a RAM disk image

If the root is set to be a ram file system, an initial file system is written into the coprocessor's memory.

After the file is written and the coprocessor booted, the init script creates a tmpfs (Linux* ram disk file system type) in the coprocessor's memory and extracts the compressed file system information into it. This image must contain all files needed to start a fully functional Linux* system.

The ram disk image is activated as the root device by calling the Linux* switch_root utility. This special utility instructs the Linux* kernel to remount the root device on the tmpfs mount directory, release all file system memory references to the old initial ram disk and start executing the new /sbin/init function.

/sbin/init performs the normal Linux* user level initialization. All the information required must already be present in the compressed cpio file.

3.1.6 Notify the host that the coprocessor is ready

The last step of any of these initializations is to notify the host that the coprocessor is ready for access. It does this by writing to its /sys/class/micnotify/notify/host_notified entry. This causes an interrupt into the host driver which updates the coprocessor's state to online.
4 Configuration

This section focuses on configuring an Intel® Xeon Phi™ coprocessor, including configuration files, kernel command line parameters, and authentication.

4.1 Configurable components

On a typical Linux* system, the installation and configuration process is performed as a series of questions posed by the system and answered by the installer/operator. Since the coprocessor does not have a file system of its own, this process is replaced by editing the configuration files and using the micctrl utility.

The configuration parameters are divided into three categories:

1. Parameters that control loading the coprocessor Linux* kernel onto its memory and initiating the boot process.
2. Parameters to define the root file system to be used on the coprocessor.
3. Parameters to configure the host end of the virtual Ethernet connection.

4.2 Configuration files

This section briefly discusses configuration file formats and the use of the Include parameter to micctrl.

4.2.1 File location and format

Configuration is controlled by per coprocessor configuration files located in the C:\Program Files\Intel\MPSS directory. Each coprocessor has an associated micN.xml configuration file, where N is the integer ID of that coprocessor (for example: mic0.xml, mic1.xml, etc.).

Each of the configuration files contains a list of parameters and their arguments.

4.3 Configuring boot parameters

The host system boots the coprocessor by injecting the Linux* kernel image and kernel command line into its memory and then instructing it to start. To perform this operation, the host system reads the configuration files and builds the kernel command line from relevant parameters. By default, the boot parameters are placed in the per-coprocessor micN.xml files, allowing each coprocessor to be configured separately. If a boot parameter is placed in the global.xml file, it will apply to all coprocessors unless overridden.
4.3.1 Power management

The PowerManagement parameter specifies the coprocessor Linux* power management settings. The system owner can specify different power management settings by editing these values. The changes take effect upon executing either `micctrl --start` or `micctrl -b`.

4.3.2 Command line

The CommandLine parameter controls what options are passed to the coprocessor when it starts.

4.3.3 Networking

The Networking parameters specify various settings such as IPAddress, HostIPAddress, Subnet, MACAddress, and HostMACAddress. These settings will not take effect until the coprocessor is restarted.

4.4 Root file system

Every Linux* system needs a root file system with a minimal set of files. Other nonessential files may be on the root or they may be on secondary mounts. Most modern Linux* OS releases assume the root file system will be large enough to hold the complete release. The coprocessor’s embedded file system currently follows the same rule.

Files on the root fall into three categories: the binaries installed with the system, the files in the /etc directory, which are used for configuring parameters uniquely to an individual system, and the set of files for the users of the system.

Intel® MPSS provides a set of configuration parameters that are used in building the root file system image. Refer to Section 4.3, and Section 6.1, for more information.

4.5 Routing

Windows* provides functionality to route network traffic between two or more networks. More information on routing can be found in the article available at https://technet.microsoft.com/en-us/library/dd469714(v=ws.11).aspx. IP routing is disabled by default.

Steps to enable routing:

1. Go to Run; type `cmd`.
2. Enter the following commands:

   Admin> sc config RemoteAccess start= auto
   Admin> sc start RemoteAccess
There is no need to modify coprocessor configuration files in order to support routing. Since *micctrl* utility currently does not support setting up the default gateway in the coprocessor, it is necessary to manually change it using the command below in the coprocessor OS.

**Note:** This operation requires SSH access to the coprocessor OS. Refer to Section 10 for instructions.

```
[micN]# route add default gw <IP Address of the Host machine>
```

### 4.6 Bridging

A network bridge is a way to connect two Ethernet segments or collision domains in a protocol independent way. It is a Link Layer device which forwards traffic between networks based on MAC addresses and is therefore also referred to as a Layer 2 device.

Two types of bridged networks are directly supported by Intel® MPSS.

#### 4.6.1 Internal bridging

Some distributed applications running on the coprocessors on a single node need to communicate between coprocessors, and perhaps with the host. An internal bridge, as presented in **Figure 2**, allows for the connection of one or more coprocessors within a single host system as a subnetwork. In this configuration, they can communicate with the host and with each other within the platform.

**Figure 2 Internal bridge network.**

Such network configuration could, for example, be used to support communication between the ranks of an MPI application that is distributed across the coprocessors and the host (however, using the IBSCIF virtual InfiniBand* HCA driver will likely provide better performance.)
4.6.2 External bridging

The external bridge configuration bridges coprocessors to an external network. This is the typical configuration required when the coprocessors are deployed in a cluster to support remote communication among them and/or processors across different compute nodes.

Figure 3 depicts a cluster in which the coprocessors on each host node are bridged to the external network. The IP addresses in such configuration can be assigned statically by the system administrator or by a DHCP server on the network, but must generally be on the same subnet.

InfiniBand* based networking is not shown in this figure. InfiniBand* based networking will usually provide significantly higher bandwidth than the IP networking supported by the Intel® MPSS Virtual Ethernet driver. Many clusters use Ethernet* networking for low bandwidth communication such as command and control and use InfiniBand* networking for high bandwidth communication as application data transfer.

Figure 3 External bridge network

To prepare for configuring this network topology, you should ensure that you have provided a large enough IP address space to accommodate the nodes of the externally bridged networks.
The micctrl utility

5 The micctrl utility

The micctrl utility can be compared to a multi-purpose toolbox for the system administrator. It provides the following categories of functionality:

- Coprocessor state control – boot and reset control.
- Configuration file initialization and propagation of values.
- Helper functions for modifying configuration parameters.

The micctrl utility requires a first argument specifying the action to perform, followed by option-specific arguments. The arguments may be followed by a list of coprocessor names, which is shown in the syntax statements as [mic card list]. For example the list may be mic1 mic3, if these are the coprocessors to control.

5.1 Coprocessor state control

The micctrl utility provides mechanisms for individual coprocessor control. micctrl controls and queries coprocessor’s state via the BootMIC WMI method.

Note: Changing the coprocessor’s state requires full administrative rights.

5.1.1 Booting the coprocessor

Command syntax:

Admin> micctrl -b [mic card list]

The coprocessor must be in the ready state in order to boot. The driver will inject the indicated Linux* image into the coprocessor’s memory and initiate the booting sequence.

5.1.2 Resetting the coprocessor

Command syntax:

Admin> micctrl -r [mic card list]

The coprocessor can be reset regardless of the state it is in. This command uses the ResetMIC WMI method. The driver will perform a soft reset on the coprocessor by setting the correct PCI mapped register.

Note: Performing a reset may result in the loss of file data that has not been flushed to a remote file.
5.1.3 Waiting for the coprocessor’s state change

Command syntax:

Admin> micctrl -w [mic card list]

The *wait* option waits for the status of the coprocessor to be either *online* or *ready*. It also allows for a brief pause to the *ready* state during the Intel® MPSS startup. It is intended for users to verify if the software stack startup, shutdown, or reset procedure is complete. This option has a built-in timeout value of 300 seconds.

5.1.4 Checking coprocessor status

Command syntax:

Admin> micctrl -s [mic card list]

The *status* option displays the status of the coprocessor.

5.2 Helper functions for configuration parameters

The following sections discuss command options for adding and removing users and groups.

5.2.1 Adding users to the coprocessor’s file system

Adding a user to the coprocessor’s file system is accomplished through the addition of a user to the MICUSERS user group on the host. See Section 10.5 for detailed instructions on creating this group and adding users to the file system.

5.2.2 Removing users from the coprocessor’s file system

Removing a user from the coprocessor’s file system is accomplished through the deletion of a user from the MICUSERS user group on the host. See Section 10.5.1 for detailed instructions on using this group to remove users from the file system.

5.2.3 Specifying the host secure shell keys

Command syntax:

Admin> micctrl --addssh <user> -f <path to public key file>
Admin> micctrl --addssh <username> <public-key>

The *--addssh* option adds the specified public key to the *authorized_keys* file in the coprocessor’s file system. The user must belong to the MICUSERS group for this action to be effective on the next boot. This command must be executed from a command prompt with full administrator privileges.
The public key file format can be OpenSSH public key format, PuTTY public key format, or PuTTY private key format (usually saved with a file extension .ppk).

Without specifying the \(-f\) option, the contents of an OpenSSH public key can be pasted directly into the command line and will be appended to the user’s \texttt{authorized\_keys} file.

The secure shell keys will not be updated until the coprocessor is rebooted using \texttt{micctrl}. It is not sufficient to power cycle the host machine.

5.3 \textbf{Other file system helper functions}

5.3.1 \textbf{Updating the compressed cpio image}

\textbf{Command syntax:}

\begin{verbatim}
Admin> micctrl -g [mic card list]
\end{verbatim}

The \(-g\) option updates the image from the parameters specified in configuration files and file lists. The new image will be used the next time the coprocessor boots.

§
6 Adding software

Typical installations are not static, and usually require the system administrator to add additional files or directories to the Intel® Xeon Phi Coprocessor root file system that is downloaded to the coprocessor.

6.1 The file system creation process

In this section we describe the process of building a root file system.

The base components of a root file system are filelists. They can be found in the installation directory under the filesystem subdirectory. There is one filelist for each coprocessor as well as a common filelist shared between all coprocessors.

Files can be added to the root file system by editing an existing filelist (for example: mic0.filelist) and adding the appropriate directives to the filelist.

There are six filelist directive types:

- **dir** <name> <perms> <uid> <gid>
- **file** <name> <source> <perms> <uid> <gid>
- **slink** <name> <to> <perms> <uid> <gid>
- **nod** <name> <perms> <uid> <gid> <type> <major> <minor>
- **pipe** <name> <perms> <uid> <gid>
- **sock** <name> <perms> <uid> <gid>

Each directive type is specific to one of six types of files available on a Linux* file system.

**Note:** perms argument indicates an octal Linux* file permission scheme, uid denotes the numeric user ID where 0 is root and 400 is micuser, and gid denotes the numeric group ID where 0 is root and 400 is micuser.

6.1.1 The dir filelist directive

The dir directive specifies a named directory to be created in the coprocessor’s file system image. The perms, uid, and gid arguments specify the directory permissions, user ID, and group ID. A typical entry is:

```
dir /tmp 0777 0 0
```

The example defines the directory /tmp to be owned by the user root and group root, and with global permissions for everybody.
6.1.2 **The file filelist directive**

The *file* directive specifies a named file to be created in the coprocessor’s file system image. The *perms*, *uid*, and *gid* arguments specify the file’s permissions, user ID, and group ID.

The source argument to *file* is relative to the location of the *filelist* itself.

For example the following *filelist* directive in C:\Program Files\Intel\MPSS\filesystem\mic0.filelist:

```plaintext
file /etc/passwd etc/passwd 644 0 0
```

will cause the /etc/passwd file to be added to the coprocessor’s file system image and populated with the contents of the file C:\Program Files\Intel\MPSS\filesystem\mic0\etc\passwd. It will be owned by user root and group root, and with global read permission and root modification permission.

6.1.3 **The slink filelist directive**

The *slink* directive specifies a named symbolic link to be created in the coprocessor’s file system image, and linked to the source. The *perms*, *uid*, and *gid* arguments specify the symbolic link’s permissions, user ID, and group ID, respectively.

A typical use of symbolic links is found in the Linux* startup scripts. The filelist associated with the configuration parameter includes the following:

```plaintext
slink/etc/rc3.d/S80sshd ../init.d/sshd 0755 0 0
```

This directs the creation of a symbolic link in the coprocessor’s file system image accessing the /etc/init.d/sshd file when /etc/rc.d/S80sshd is accessed.

6.1.4 **The nod filelist directive**

The *nod* directive specifies a named device node of a specified type to be created in the coprocessor’s file system image. *Type* must be either the character *b* for block device or *c* for character device. The arguments major and minor must be integer values defining the correct values of the node. The *perms*, *uid*, and *gid* arguments specify the device node’s permissions, user ID, and group ID, respectively.

Most device nodes are created dynamically by a device driver. However, some legacy devices still require a hard-coded entry.

For example: The filelist for BaseDir includes the following entry, which specifies the creation of a device node for the console:

```plaintext
nod/dev/console 0600 0 0 c 5 1
```
6.1.5 The pipe filelist directive

The pipe directive specifies a named pipe device file to be created in the coprocessor’s file system image. The perms, uid, and gid arguments specify the pipe’s permissions, user ID, and group ID, respectively.

6.1.6 The sock filelist directive

The sock directive specifies a named socket device file to be created in the coprocessor’s file system image. The perms, uid, and gid arguments specify the socket’s permissions, user ID, and group ID, respectively.

6.2 Adding persistent files to the coprocessor’s file system

Files added to the coprocessor’s file system during normal use are not stored persistently by default. There are several lists of files that are used to generate persistent file systems. They are located in C:\Program Files\Intel\MPSS\filesystem.

The \common directory and corresponding common.filelist can be used by the micctrl utility to add files and directories to the file system that will be present on all coprocessors. Create \common directory and common.filelist to use this feature.

Finally, there is a directory mic0, mic1, etc., for each coprocessor installed in the system. These (and corresponding micN.filelist) are used to add files and folders unique to an individual coprocessor’s file system.

To add a file to the coprocessor’s file system, the format is:

file /path/on/filesystem path/on/host PERM UID GROUP

where PERM indicates an octal Linux* file permission for the file. UID denotes the numeric user ID where 0 is root and 400 is micuser. GROUP denotes the numeric group ID where 0 is root and 400 is micuser.

To add a directory to the file system, the format is:

dir /path/on/filesystem PERM UID GROUP

with PERM, UID, and GROUP defined similarly as above.

Once the modifications have been made to the filelist, the necessary files must be placed into the corresponding directories. The following example adds the directory permanent to the root of the file system that contains a file foo.txt.

1. Append the following lines to mic0.filelist:

   dir /permanent 755 400 400
   file /permanent/foo.txt permanent/foo.txt 644 400 400
2. Now, create the following directory and place the file *foo.txt* in it:

\`C:\Program Files\Intel\MPSS\filesystem\mic0\permanent\`

3. Execute the following sequence to restart the coprocessor:

```
Admin> micctrl –stop
Admin> micctrl --start
```

After restarting the coprocessor, the directory and file will be in the coprocessor file system persistently.

6.3 Creating the download image file

The download image file is created by processing the common *filelist* for and then the *filelist* for a coprocessor, in that order.

When the *filelists* are completely processed, *micctrl -g* will create a cpio entry for the file and append it to the *micN.image* file, where N is the numeral indicator for that device.

6.4 Adding files to the root file system

Adding a file to the root file system can be done by adding an entry to some existing *filelist*, indicating the location of the file.

6.4.1 Adding files by copying

When adding a file to an existing *filelist*, the first decision is whether the file should be accessible by all coprocessors or only a particular one. If it is required for all coprocessors to have access, then copy the file to a location under the *common* directory and amend its *filelist*. Otherwise, copy the file to the directory *micN* for coprocessor N. Then update the corresponding *filelist*.

If a directory had to be created for the added file, insert the appropriate *dir* entry prior to the new file entry.
7 Intel® Xeon Phi™ coprocessor tools and utilities

7.1 Coprocessor information tool: micinfo

The micinfo command displays information about the coprocessors installed in the system along with relevant details about the host system, micro-OS and the drivers. The default installation location for the micinfo.exe tool is the C:\Program Files\Intel\MPSS\bin directory.

7.1.1 Simple method

The following is the simplest way to execute micinfo:

User> micinfo

7.1.2 For advanced users

Command syntax:

User> micinfo [OPTIONS]

OPTIONS:

-help: Display command help.
-version: Display the tool version.
-listDevices: List all coprocessors detected.
-deviceInfo <deviceNum> [-group <groupName>]: Displays information about the user-specified coprocessor (determined by <deviceNum>). User may additionally specify the type of information with -group <groupName> option.

Valid values for <groupName> are:

- Versions: Show Flash and uOS versions.
- Board: Show coprocessor PCIe board related information.
- Core: Show number of cores, voltage and frequency.
- Thermal: Show fan and thermal related data.
- GDDR: Show device memory related information.
7.2 The coprocessor platform status panel: micsmc

*micsmc* is the coprocessor Platform Status Panel. The *micsmc* tool can function in two modes: a graphical user interface (GUI) mode and a command-line interface (CLI) mode. The GUI mode provides real-time monitoring of all detected coprocessors installed in the system. The CLI mode produces a snap-shot view of the status, which allows it to be used in cluster scripting applications. The *micsmc* tool, among other features, monitors core utilization, temperature, memory usage, power usage statistics, and error logs.

The default installation location for *micsmc* is `C:\Program Files\Intel\MPSS\bin`.

The *micsmc* tool is based on the work of the Qwt project ([http://qwt.sf.net](http://qwt.sf.net)).

The *Status Panel User Guide* is available in all supported languages, in PDF and HTML formats, by default in:

`C:\Program Files\Intel\MPSS\docs\micmgmt`

7.3 The coprocessor verification tool: miccheck

The *miccheck* utility is used to verify the configuration and current status of the software stack. It performs sanity checks on a host system with coprocessor(s) installed, by running a suite of diagnostic tests. The default behavior is to run all enabled tests on the host system first, and then on each coprocessor in turn.

For detailed information about *miccheck*, refer to the help option of the program:

```
User> miccheck --help
```

7.4 The coprocessor RAS tool: micras

*micras* is the application running on the Host system that collects and logs RAS events generated by the coprocessor(s). This tool is also responsible for handling test and repair by kicking the coprocessor into maintenance mode upon the detection of an uncorrectable or fatal RAS event. It runs as a Windows* service. The default installation location is `C:\Program Files\Intel\MPSS\service`.

*micras* logs messages into the *micras.log* file located under `C:\Program Files\Intel\MPSS\service`. The log messages include but are not limited to:

- MCA events including both correctable and uncorrectable events.
- Coprocessor reset, maintenance mode test or repair messages.
- RAS daemon software operation messages.
An example of the RAS log entry is shown below:

Tue Mar 5 16:24:29 2013 MICRAS ERROR : Card 2: failed getting card mode
(Timestamp) (Severity level) (Message body)

7.4.1 Simple method

micras is installed as a Windows* service.

The following is the simplest way to start micras service:

Admin> micctrl --start
Admin> net start micras

To stop micras service, execute the following command:

Admin> net stop micras

7.4.1.1 Configure micras service to start when Windows* boots (optional)

To configure the micras service (RAS) to start when the Windows* OS boots, follow the instructions:

1) In a command window, type services.msc, and press Enter.
2) In Services, right-click Intel(R) Xeon Phi(TM) coprocessor Reliability, Availability Service, and click Properties.
3) On the General tab, in Startup type, select Automatic, and click OK.

7.4.2 For advanced users

The micras tool can be used in these various ways. See the detailed usage below:

Use net start micras to start micras service and net stop micras to stop micras service from a Windows* command prompt.

Command syntax:

User> micras [OPTIONS]

OPTIONS:

-help Display command help information.

-maint Enable maintenance mode for error test and repair.

-loglevel [loggingLevel] set the level of detail that gets logged with the micras tool. The accepted levels are from 1 to 15.

It is a 4-bit representation, where bits 0 - 3 denote the following:

Bit 0 – Enables Informational Messages
**Bit 1** – Enables Warning Messages  
**Bit 2** – Enables Error Messages  
**Bit 3** – Enables Critical Messages  
**Default** – all messages on.

The severity level of *micras* log messages is mostly aligned with the standard RFC 5424 syslog severity level. Currently, there are four severity levels available (refer to Table 2).

If *micras* is executed with no arguments, it runs at the console prompt, connects to devices, and waits for errors.

Use *Ctrl-C* to exit *micras* and return to the console prompt.

Use *-maint* option to enable maintenance mode for error test and repair.

Use *-help* to show the help info.

**Table 2 micras severity levels**

<table>
<thead>
<tr>
<th>Severity</th>
<th>Description</th>
<th>General Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Informational messages</td>
<td>Normal operational messages, for information purposes (such as reporting).</td>
<td>No action required.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning conditions</td>
<td>Warning messages. Not an error, but an indication that an error might occur if action is not taken.</td>
<td>No immediate action required.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error conditions</td>
<td>Non-urgent failures. The failure might be recovered by software itself.</td>
<td>Inform your system administrator.</td>
</tr>
<tr>
<td>CRITICAL</td>
<td>Critical conditions</td>
<td>Critical conditions that should be corrected immediately. <em>micras</em> software has some test and repair capability built-in. However, some critical conditions will require admin’s involvement like removing/replacing components.</td>
<td>Immediate action required for some error conditions</td>
</tr>
</tbody>
</table>

**7.5 The coprocessor utility: micnativeloadex**

The *micnativeloadex* utility will copy a coprocessor native binary to a specified coprocessor and execute it. The utility automatically checks library dependencies for
the application, and copies them to the device prior to execution provided they can be
found in the default search path (set using the SINK_LD_LIBRARY_PATH environment
variable). This simplifies running the coprocessor native applications since the utility
automatically copies the required dependencies.

If the application has any library dependencies, the SINK_LD_LIBRARY_PATH
environment variable must be set to include those dependencies. This environment
variable works just like LD_LIBRARY_PATH for standard applications. To help
determine the required libraries, execute mcnativeloadex with the -l command line
option. This will display the list of dependencies and indicate which ones have been
found. Any dependencies not found will likely need to be included in the
SINK_LD_LIBRARY_PATH.

In addition, the utility can also redirect output from the application running remotely
on the coprocessor back to the local console. This feature is enabled by default, but
can be disabled with a command line option. For further details on command line
options, refer to its help section:

    User> mcnativeloadex -h

§
8 **Host file system share**

This chapter demonstrates how to set up an NFS share on Windows* Server 2012, Windows* Server 2012 R2 and Windows* Server 2016.

*Note:* Feature described in this section is not supported on Windows* 8.1 and Windows* 10.

### 8.1 Creating an NFS share

Section 8.1.1 through 8.1.3 use the Server Manager user interface to set up an NFS share on Windows* Server 2012, Windows* Server 2012 R2 and Windows* Server 2016. For an alternative method using PowerShell* cmdlets, skip to Section 8.1.4.

#### 8.1.1 Adding the NFS server

1. Start **Server Manager**. On the dashboard, click **Add Roles and Features** on the **Manage** menu.

2. In the **Add Roles and Features Wizard**, click **Installation type** in the left column. Select **Role-based or feature-based installation**.
3. Click **Server Selection** and **Select a server from the server pool**. Select the server.

4. Click **Server Roles**. Under **Roles**, expand **File and Storage Services**, then expand **File and iSCSI Services**. Select **Server for NFS**.
5. A confirmation dialog box will appear. Click the **Add Features** button.

![Add Roles and Features Wizard](image1)

6. Returning to the **Add Roles and Features Wizard**, select **Confirmation** in the left column. Click the **Install** button.

![Add Roles and Features Wizard](image2)
7. In the **Results** pane, confirm that the installation was successful.

![Add Roles and Features Wizard](image)

8.1.2 **Provisioning a folder for the NFS share**

1. In Server Manager, click **File and Storage Services** in the left column of the Dashboard.

![Server Manager Dashboard](image)
2. Select the desired server from the list. Click **Shares** in the left column.

![Server Manager with Shares selected]

3. Click **To create a file share, start the New Share Wizard**.

![Server Manager with New Share Wizard option highlighted]

4. In the **New Share Wizard**, click **Select Profile** and select **NFS Share - Quick**.

![New Share Wizard with NFS Share profile selected]
5. Click **Share Location** and select the desired server from the list. Specify a path for the share.

![New Share Wizard](image1)

6. Click **Share Name** and specify a name for the share (click **OK** if prompted to create the share location directory).

![New Share Wizard](image2)
7. Click **Authentication**. Select **No server authentication**, **Enable unmapped user access**, select **Allow unmapped user access by UID/GID**.

8.1.3 **Specifying the NFS share permissions**

1. Click **Share Permissions** and click **Add**. In the **Add Permissions** dialog box, change the **Share Permissions** field for the **All Machines** group to **No Access**. This prevents unspecified hosts from accessing the NFS share. Click **Add**.
2. Return to the **New Share Wizard** and click **Add**. Now, select **Host** in the **Add Permissions** dialog box. Specify a host and set its **Share Permissions** field to **Read/Write**. Select **Allow root access**. Click **Add**.

![New Share Wizard](image1)

3. Return to the **New Share Wizard**. Review the **Share Permissions** settings. Verify the Host has **Read/Write** permissions and **Root Access** is **Allowed**.

![New Share Wizard](image2)
4. In the left column, click **Permissions** then click **Confirmation**. Confirm your settings and click **Create**.

![New Share Wizard](image)

5. Confirm your results and close **New Share Wizard** and **Server Manager**.

![New Share Wizard](image)

6. Proceed to **Section 8.2**.

**8.1.4 Using Powershell* cmdlets to create an NFS share (optional)**

Alternatively, the tasks in previous sections can be accomplished using the built-in PowerShell* NFS cmdlets in Windows* Server 2012, Windows* Server 2012 R2 and Windows* Server 2016, as shown below:

*Note:* The backslash \ character, placed at the end of a line, is used to indicate continuation of a command on the next line. It is not part of the command itself.

1. Add the NFS server:

   User> Add-WindowsFeature FS-NFS-Service

2. Provision the directory to be shared:

   User> New-Item C: \MY_NFS_TEST -type directory
User> New-NfsShare -Name nfs-tests \  
-Path C:\MY_NFS_TEST -EnableUnmappedAccess $True \  
-Permission no-access -Authentication sys

3. Grant read/write and root access permissions for Host 192.168.1.100:
User> Grant-NfsSharePermission -Name nfs-tests  
-Permission readwrite -ClientName 192.168.1.100 \  
-ClientType Host -AllowRootAccess $True

4. Confirm share permission settings:
User> Get-NfsSharePermission -Name nfs-tests

5. Proceed to the next section.

8.2 Mounting the NFS share

Note: Refer to Section 10.1 for information on downloading and installing PuTTY tools.

1. Change to the directory where PuTTY tools are installed:
User> cd C:\Program Files\Intel\MPSS\bin

2. Log in to the coprocessor as root:
User> putty.exe -ssh -i \  
(PATH_TO_PRIVATE_KEY) root@192.168.1.100

Note: If prompted for a password, see Section 10.3 on setting up user SSH keys for the root user.

3. On the coprocessor add a directory with the same name as the NFS share:
[mic0]# mkdir /tmp/nfs-tests

Note: Default IP address of the coprocessor is 192.168.N+1.100 where N is the integer ID of that coprocessor (for example: mic0, mic1, etc.).

4. Mount the NFS share from the coprocessor. The nolock option is required on this step:
[mic0]# mount -t nfs -o nolock \  
192.168.1.99:/nfs-tests /tmp/nfs-tests

5. Navigate to the directory that you created on the coprocessor. Verify that the contents of the NFS share are visible.
[mic0]# cd /tmp/nfs-tests

§
9 CIFS share on windows*

9.1 Mounting CIFS shares with micctrl

Command Syntax:

```
micctrl --addcifs <server> <sharename> <card mountpoint>
```

`micctrl --addcifs` option prepares the setup required for mounting. `micctrl` updates the configuration file, `filelist` and `fstab` with minimum required settings to mount as a guest user. Coprocessors need to be restarted to access the mounted share.

To mount a CIFS share, first grant share permissions to **Everyone** and disable **password protected sharing**.

1. Open **Network and Sharing Center**, choose **Change advanced sharing settings**, expand **All networks** and enable the **Turn off password protected sharing** option.

2. Choose a folder you wish to share and grant appropriate permissions to **Everyone**.

   a. Open properties of the folder you wish to share, select the **Security** tab and click **Edit**... to change the folder permissions.
b. Click **Add...**, type **Everyone** in the **Enter the object names to select** field and click **Ok**.

c. Grant appropriate permissions to the **Everyone** group, which should now be visible in the **Permissions** window.

3. Navigate to the properties of the folder you wish to share, choose the **Sharing** tab and click **Share**. Add **Everyone** to the share group, assign appropriate permissions and select **Share**.
4. Click **Advanced Sharing** in the **Sharing** tab and select **Share this folder**.

![Advanced Sharing](image)

5. Navigate to **Permissions** and assign appropriate permissions to **Everyone**.

![Permissions](image)

**Note:** You have to enable share permissions in both **Share** and **Advanced Sharing**. The current mount utils on coprocessor do not support nested host share paths.

6. Run the `micctrl --addcifs` command. You can provide a space-separated list of coprocessors’ names to enable the mount on the specified devices. If no coprocessors are specified, the mount will be enabled on all coprocessors in
the system. Running this command should add a \textit{<CIFS>} entry in the \texttt{micN.xml} file. Restart the coprocessors to apply the new settings.

**Example:**

```
micctrl --addcifs host winshare /mnt/wincifsshare mic0 mic2
micctrl --addcifs 192.168.1.99 winshare /mnt/wincifsshare
```

The last step updates the \texttt{micN.filelist}, corresponding \texttt{fstab} and mounts the share on boot. You can access the mounted share using PuTTY, which would need prior setup of ssh using the \texttt{micctrl --addssh} option.

You can manually edit the \texttt{micN.xml} configuration file if you want to modify a mounted share, it is the same as adding additional mount options. \textit{<coprocessor mount point>} is used as a unique identifier when there are multiple CIFS shares. For example, there should be one \textit{<CIFS>} entry per mount point in the configuration file. You may wish to delete the old settings from \texttt{fstab} to avoid conflicts. You have to restart the coprocessor to access the new mounted share.

You have the option to manually \texttt{mount} and \texttt{umount} password protected shares as the root user on the coprocessor.

### 9.1.1 Mounting a temporary CIFS share

Users can issue the following command on the coprocessor to temporarily mount a share. This operation allows to check whether the share is functioning correctly and that users on the coprocessor have appropriate permissions to open it.

```
[micN]$ mount -t cifs \texttt{<SHAREPATH>} \texttt{<MOUNTPOINT>} -o guest
```

\texttt{<SHAREPATH>} indicates a network addressable CIFS share and \texttt{<MOUNTPOINT>} is the directory in the coprocessor’s file system over which to mount the share.

### 9.1.2 Troubleshooting

When accessing the share files on the coprocessor, you could encounter \textit{cannot allocate memory} error. This error can be avoided by maximizing the throughput for file sharing and/or network applications in the registry.

- Set the value of the registry key below to 2 or 3 (for large file transfers):

  \texttt{HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters\size}

- You can also optionally set the value of the registry key below to 1:

  \texttt{HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\LargeSystemCache}

Reboot your host or restart the \texttt{Server} service in \texttt{services.msc} after introducing the changes.

**References:**

- [http://support.microsoft.com/kb/232271](http://support.microsoft.com/kb/232271)
9.2 Configuring a CIFS share with a bridge

**Note:** The *IP addr* should be specified instead of *host*, when bridging is setup.
1. Configure a bridge,
2. Change the IP address in the xml files
3. Add a CIFS share with *micctrl*
4. See the xml file.

9.3 Umounting a CIFS share with *micctrl*

**Command Syntax:**

```
micctrl --removecifs <card mount point> [mic card list]
```

*micctrl --removecifs* option prepares the setup required for umount. It doesn’t do the actual umount but instead cleans up the configuration, *filelist* and *fstab* settings updated in *--addcifs* step. You have to restart the coprocessors for removing the mounted share.

1. Run the *micctrl --removecifs* command. You can use this option to clean up the CIFS settings on specified coprocessors by providing a space-separated list of coprocessors’ names. If the list is not provided, *micctrl --removecifs* will attempt to clean up the settings on all available coprocessors in the system.

**Example:**

```
micctrl --removecifs /mnt/wincifsshare mic0 mic2
```

2. Restart the coprocessors. The mounted share will be removed on boot.

§
10 Interacting with the Intel® Xeon Phi™ coprocessor

It is possible to interact with the coprocessor using standard SSH tools. For the purposes of this section it is assumed that the PuTTY tools will be used. The following sections describe how to download and use this software.

10.1 External tools

Some Intel® tools require that PuTTY and PuTTYgen be installed. It is possible to download them from:
Install PuTTY and PuTTYgen to the same location as the external tools, the default folder is C:\Program Files\Intel\MPSS\bin.

10.2 Special user account micuser

The default users in each coprocessor file system are root and micuser. Micuser is a special user account, reserved exclusively for COI offload applications. Users are not permitted to login as micuser.

The special user account micuser must not be confused with the MICUSERS user group. Users must be listed in the MICUSERS group in order to log in to the coprocessor. For more information on adding and deleting users from the MICUSERS group, see Section 10.5 and 10.5.1.

10.3 SSH access to the coprocessor

Follow the chosen SSH key tool documentation to generate public and private SSH keys. This section shows instructions for PuTTY tools.
Generating public and private SSH keys:

1. Launch the puttygen.exe application and click Generate. After following the on-screen instructions the following screen is displayed:

![PuTTY Key Generator](image)

2. Click Save public key to save the public key to disk in SSH-2 format. For example: save the key with the name publicKeys.pub. Click Save private key to save the private key in a secure location, inaccessible by other users. This key will be used later to authenticate the connection with the coprocessor. Save the key with a passphrase or, click Yes to ignore the warning. Close the PuTTYgen tool.

3. Open a command prompt as an Administrator and run the command:

```
Admin> micctrl -addssh <username> -f <path-to-key-file>
```

<username> is the actual username to link with this SSH key and <path-to-key-file> is the full path to the public key file (saved during key generation).

Alternative Procedure for Steps 2 & 3:

Copy the entire text in Public key for pasting into OpenSSH authorized_keys file field into the clipboard. Save the private key as described in Step 2 above. Close the PuTTYgen tool.

Open a command prompt as an Administrator. Run the command:

```
Admin> micctrl -addssh <username> <public-key>
```
where <username> is the actual username to link with this SSH key and <public-key> is the key that was copied to the clipboard from the PuTTYgen tool. Paste the <public-key> directly into the command.

4. Restart the coprocessors using the micctrl utility (a host reboot is not sufficient to generate a new file system). Use standard SSH tools to interact with the specified coprocessor once its state changes to online. Note the user which now has SSH privileges, this username will be used to interact with the coprocessor.

**Note:** The default accounts on coprocessor are root and micuser. The special user account micuser is used exclusively for COI offload applications. Users are not permitted to login as micuser.

**Establishing connection with a coprocessor:**

1. Once Intel® MPSS has started, start putty.exe. In the **Category** box, expand **Connection -> SSH** and select **Auth**. In the **Private key file for authentication** field, browse to your privateKey.ppk (saved during key generation).

2. In the **Host Name (or IP address)** field, enter 192.168.1.100 and then click **Open**. This operation opens an SSH session to coprocessor mic0.

   You can also click **Save** to save the SSH session (you can choose a default session name or create a new one). This allows easy reconnection to the coprocessor without specifying private keys each time.
3. When the SSH session opens to the coprocessor, enter your username, if previous steps were completed correctly you will not be asked for password. You should now have a terminal session set up to the file system on the coprocessor.

After completing these steps, a program such as WinSCP can be used to transfer files to and from the coprocessor.

10.4 Transferring files to and from the coprocessor

This section describes using the WinSCP tool (version 5.5 shown) to transfer files to and from the coprocessor. WinSCP is open source and can be downloaded from http://winscp.net/.

1. Launch WinSCP and select File Protocol: SCP. Enter 192.168.1.100 in the Host Name field and your username in the User Name field. Do not enter a password. Click Advanced... to display the Advanced Site Settings dialog.
2. In the **Advanced Site Settings** dialog, select **SSH -> Authentication**. In the **Private key file** field, enter the full path to the private key file generated above (or click [...] to browse to the file). Click **OK** to return to the previous dialog.

3. Optionally, click **Save As...** to save the new site to the tree in the left pane. Click **Login** to use the tool.
10.5 User groups

The default users for the coprocessor are root and micuser. The special user account micuser is used exclusively for COI offload applications. Users are not permitted to login as micuser. The administrator can add additional users. Any new user for the coprocessor must have a corresponding local or domain user account.

1. Click the Windows* Start button. Enter Run in the Search Programs and Files search box and press Enter. The Run window will appear.

2. Enter lusrmgr.msc in the Open drop down box of the Run window and press Enter.

3. Right click Users from the left pane and select New User from the pop up menu to add a new user.

4. Right click Groups from the left pane, and select New Group. Type MICUSERS in the box titled Group name. Click Add under the Members box, and type in the name of the User you created in Step 3. After adding the user, select Create to create the MICUSERS group.

Note: Only users listed in the MICUSERS group are permitted to log in to the coprocessor. After adding or deleting users in the MICUSERS group, the coprocessor must be restarted.

10.5.1 Deleting users from the micusers group

The administrator can remove users from the MICUSERS group, as follows:
1. Click the Windows* Start button. Enter Run in the Search Programs and Files textbox and press Enter. The Run window will appear.

2. Enter lusrmgr.msc in the Open drop down box and press Enter.

3. Double click the Groups selection in the center panel.

4. Right click the MICUSERS group in the middle pane and select Properties. In the Member pane select the user name to delete from this group. Click Remove. Click OK.

5. Once all users have been removed from the group, click OK. The newly-removed users will no longer be available in the group once the coprocessor is restarted.

**Note:** Only users listed in the MICUSERS group are permitted to log in to the coprocessor. After adding or deleting users in the MICUSERS group, the coprocessor must be restarted.

### 10.6 Running the COI tutorials (optional)

After installation is complete, it is possible to run the COI tutorials.

See C:\Program Files\Intel\MPSS\sdk\tutorials\coi\README_Windows_1.txt for instructions.

Intel® Coprocessor Offload Infrastructure (Intel® COI) provides a set of APIs to simplify development of tools and other applications using offload and reverse accelerator models.

### 10.7 Running the MYO tutorials (optional)

After installation is complete, it is possible to run the MYO tutorials.

See C:\Program Files\Intel\MPSS\sdk\tutorials\myo\README.txt for instructions.

MYO is a library and API to support virtual shared memory between processes on a host process and coprocessors. MYO is supplementary to other Intel® Xeon Phi™ coprocessor hardware and software, and is intended for researchers and advanced users.

### 10.8 Running the SCIF tutorials (optional)

After installation is complete, it is possible to run the SCIF tutorials.

See C:\Program Files\Intel\MPSS\sdk\tutorials\scif\README.txt for instructions.

SCIF provides a mechanism for communication between the components of a distributed application. It is intended for tools and application developers.
10.9 Running the MicMgmt tutorials (optional)

After installation is complete, it is possible to run the MicMgmt tutorials.

See C:\Program Files\Intel\MPSS\sdk\tutorials\micmgmt\readme.txt for instructions.

The MicMgmt API aids developing custom cluster functionality to access and control hardware registers and parameters of coprocessors.

10.10 Troubleshooting

This section describes several methods of troubleshooting and recovery during the following events:

1. The coprocessor times out during the boot process
2. The coprocessor will not start

The micctrl utility logs information related to events in the Windows* Event Viewer. To open the event viewer, click the Start button and type Event Viewer. In Event Viewer, in the left column, expand Applications and Services and double-click MPSSLog.

This will display the following window:

![Event Viewer Window](image)

This event log should be checked in case of any observable failure starting or stopping coprocessors. Failures include timeouts and errors running micctrl –start, micctrl –stop, micctrl –r, and micctrl –b.

The event log for the coprocessors can be saved to a file from the command line with the following command (run as administrator):

Admin> wevtutil qe /f:text mpsslog > output.txt
Currently, it is recommended to use DebugView to obtain troubleshooting information related to the kernel mode drivers. DebugView can be obtained from the following website:

http://download.sysinternals.com/files/DebugView.zip

Once the zip file has been extracted, run Dbgview.exe as Administrator. From the Capture menu, make sure to select Capture Global Win32, Capture Kernel, and Enable Verbose Kernel Output. The result should look like the following image:

![DebugView capture](image)

Certain host kernel mode driver events will print a message to this window. To capture the output, click File and Save As, and enter the desired filename. DebugView is useful in cases where the micctrl utility gives commands to boot the coprocessor but the host driver fails the boot request.

Some errors can be caused by incorrect filelist specifications. See Section 6, for more details on the filelist syntax. If the filelist is incorrect, the micctrl utility cannot boot the coprocessors. To verify that the filelist syntax is correct before initiating the boot process, run micctrl -g to see errors listed on the command line. Once the micctrl utility has attempted to boot the coprocessors, the same error messages will be listed in the Event Viewer log.
11 Installing Intel® Xeon Phi™ coprocessor Performance Workloads (optional)

11.1 Requirements

1. Intel® Parallel Studio XE Composer Edition Requirements

   There are two options to installing the Intel® Parallel Studio XE Composer Edition requirements. The first option is to install full software suit.

   If the full installation is not available, two packages can be used instead. The required shared object libraries can be installed via the Intel® Parallel Studio XE Composer Edition redistributable package, freely distributed on the web at:


   The distributed archive contains an MSI package which will set up the environment and install the required dependencies.

   Another requirement, besides the shared object libraries, is the MKL Linpack benchmark. Similarly, it is freely distributed on the web at:


   This download is a zip file that can be unpacked anywhere, but the environment variable MKLROOT must point to the top level directory of the extracted package. For instance, if the user extracted the zip file into their home directory, they should set MKLROOT as follows:

   User> set MKLROOT=%HOMEPATH%\w_mklb_p_2017.2.017

   If MKLROOT is set in the user's shell environment at run time, micprun will be able to locate the linpack binaries. Note, that the downloaded version of linpack may be newer than shown above (17.02), the MKLROOT variable should reflect that.

2. MATPLOTLIB Requirements

   The micpplot and micprun applications use the MATPLOTLIB Python module to plot performance statistics. The micprun application only creates plots when verbosity is set to two or higher, and only requires MATPLOTLIB for this use case. MATPLOTLIB must be installed in order to create plots. It can be downloaded from matplotlib.sourceforge.net
3. PuTTY Requirements

The PuTTY command line utilities `pscp.exe` and `plink.exe` must be installed to a location referenced by the user's PATH environment variable. Refer to Section 10.1 for information on downloading PuTTY tools.

**Note:** The user ID and SSH key required to log in to the coprocessor can be set with the environment variables `INTEL_MPSS_USER` and `INTEL_MPSS_SSH_KEY` respectively.

**Note:** `INTEL_MPSS_SSH_KEY` should be the path to the PuTTY SSH key file.

4. User Access Requirements

Since `micperf` transfers files to the coprocessor, additional steps must be performed for new users to gain access to it:

a) By default, `micperf` uses the host user name to access the coprocessor. This user must be added to the MICUSERS group (refer to Section 10.5 for steps to create the MICUSERS group and add users to the file system).

b) After the user has been added to the MICUSERS group, refer to Section 10.4 to grant the user access to the coprocessor.

11.2 Distributed files

The `micperf` software is part of the Intel® MPSS EXE package and will be installed along with the rest of the software stack by default. The Intel® MPSS executable installs the `micperf` files to the following directories:

- **Benchmark source code:** `C:\Program Files\Intel\MPSS\sdk\tutorials\micperf`
- **Python source distribution:** `C:\Program Files\Intel\MPSS\sdk\micperf\micp`
- **Benchmark binaries:** `C:\Program Files\Intel\MPSS\sdk\micperf\libexec`
- **Reference data:** `C:\Program Files\Intel\MPSS\sdk\micperf\data`

11.3 micp python installation

**Note:** This section requires that Python be installed first. The script below requires Python version 2.7.6. You can download it from [http://www.python.org/download/releases/2.7.6/](http://www.python.org/download/releases/2.7.6/)

Once the Intel® MPSS has been installed, an additional step is required to access the `micp` Python package: either install it to your global Python site packages, or set up your environment to use the `micp` package from the installed location.

To install into the Python site packages, open a Windows* command prompt as administrator and run the following two commands:

```
Admin> cd C:\Program Files\Intel\MPSS\sdk\micperf\micp
Admin> setup.py install
```
Installing Intel® Xeon Phi™ coprocessor Performance Workloads (optional)

This method provides access to the micp package and executable scripts by all non-admin users who use the same Python version as the administrator. If Python is in the default location and uses a standard configuration, setup.py installs the micp package to the directories:

- $C:\PythonXY\Scripts$
- $C:\PythonXY\Lib\site-packages\micp$

An intermediate product of running setup.py install is the creation of the directory:

- $C:\Program Files\Intel\MPSS\sdk\micperf\micp\build$

None of the products of running setup.py discussed above will be removed by uninstalling the Intel® MPSS. The installation with setup.py uses Python's distutils module, which does not support uninstall.

§
12 Important considerations

12.1 Disabling and enabling the Memory Control Group (cgroup)

The memory Control Group is disabled by default in this release, but it can be enabled in the global.xml file. Enabling the memory cgroup decreases the amount of memory available to applications on the coprocessor by about 120MB.

PREREQUISITES:

The following settings can only be modified by the admin user.

- To enable the memory cgroup, remove the <cgroup_disable>memory</cgroup_disable> line from the <CommandLine> element in C:\Program Files\Intel\MPSS\global.xml, and then restart the coprocessors.

1. In C:\Program Files\Intel\MPSS\global.xml:
   <CommandLine>
   <clocksource>tsc</clocksource>
   <highres>off</highres>
   <mce>on</mce>
   <!-- Remove the quiet element for more verbose boot logging-->
   <quiet />
   <cgroup_disable>memory</cgroup_disable>
   </CommandLine>

2. Restart the coprocessors:
   Admin> micctrl -stop
   Admin> micctrl --start

- To disable the memory cgroup, nest the <cgroup_disable>memory</cgroup_disable> line within the <CommandLine> element in C:\Program Files\Intel\MPSS\global.xml, and restart the coprocessors:

1. In C:\Program Files\Intel\MPSS\global.xml:
   <CommandLine>
   <clocksource>tsc</clocksource>
   <highres>off</highres>
   <mce>on</mce>
   <!-- Remove the quiet element for more verbose boot logging-->
   <quiet />
   <cgroup_disable>memory</cgroup_disable>
   </CommandLine>
2. Restart the coprocessors:
   Admin> micctrl -stop
   Admin> micctrl --start

12.2 Enabling Windows* MIC GDB debugging for offload processes

For Supporting the use of the Windows* MIC GDB debugger through Microsoft* Visual Studio*, environment variables must be set in order to allow the debugger to have the files needed to attach to the coprocessor-side offload processes. To do this, execute the following steps:

1. Close Microsoft Visual Studio*, if you have not already done so.
2. Go to Control Panel->System->Advanced system settings.
3. In the Advanced tab, click Environment Variables...
4. Click New. Set Variable name to AMPLXE_COI_DEBUG_SUPPORT and Variable value to TRUE.
5. Click New. Set Variable name to MYO_WATCHDOG_MONITOR and Variable value to -1.
6. Reopen Microsoft Visual Studio*, so that the new environment will be picked up.

Upon executing steps 1-6, the necessary debug files will be created for the Windows* MIC GDB debugger to find and attach to the coprocessor-side offload processes for debugging issues in offload programs.

12.3 Enabling Windows* MIC debugging for MYO applications

Background:

The MYO WATCHDOG MONITOR is a feature of the MYO runtime library, which monitors whether the host-side and coprocessor-side are still operating and have not suffered a fatal error. If such event occurs on one peer (host or coprocessor), the other peer (coprocessors or host respectively) will abort with an error message. The MYO WATCHDOG MONITOR performs this task by making sure messages are being sent and received at a minimum rate (default 1 Hz), in order to communicate that the peers are still operating properly.

*Note:* When a MYO application is stopped at a breakpoint in a debugger, the MYO WATCHDOG MONITOR assumes that the host or coprocessor side of the MYO application has suffered a fatal error and will abort, thus preventing debugging of the MYO application.
Important considerations

To debug MYO applications for this release, the MYO WATCHDOG MONITOR must be turned off prior to starting the MYO application on the host. To do so, set the environment variable: *MYO_WATCHDOG_MONITOR* to `-1`

12.4 Copying RPMs to the coprocessor using `pscp`

In the following steps, *coreutils* and *libgmp* rpm files will be copied to the mic0 coprocessor and then installed. These steps can be used to copy and install any rpm file.

1. Go to the [Intel® Developer Zone](https://software.intel.com) website (Intel® DZ).
2. Download the *mpss-*<version number>*-k1om.tar* file from the Software for Coprocessor OS link associated with your Intel® MPSS release.
3. Extract the *mpss-*<version number>*-k1om.tar* file using a tool such as 7-Zip or WinZip.
4. In a Windows* command window, navigate to the extracted folder.
   
   ```shell
   User> cd mpss-<version number>/klom
   ```
5. Copy the *coreutils* and *libgmp* RPMs to the coprocessor with a secure copy tool such as `pscp.exe`.
   
   ```shell
   User> pscp -scp -i <PATH_TO_PRIVATE_KEY> coreutils*.rpm /root@192.168.1.100:/tmp
   User> pscp -scp -i <PATH_TO_PRIVATE_KEY> libgmp*.rpm /root@192.168.1.100:/tmp
   ```
6. SSH to the coprocessor
   
   ```shell
   User> putty.exe -ssh -i <PATH_TO_PRIVATE_KEY> \root@192.168.1.100
   ```
7. Install the *coreutils* and *libgmp* RPMs.
   
   ```shell
   [mic0]# rpm -ihv /tmp/coreutils*.rpm /tmp/libgmp*.rpm
   ```
   It is also possible to use `zypper` to install the packages:
   
   ```shell
   [mic0]# zypper --no-gpg-checks install \ /tmp/<rpm_package_name>
   ```
8. Repeat Steps 4-6 for the remaining coprocessors.

If the RPMs are to persist in the coprocessors’ file system, see [Section 6.2](#).

§
13 Related documentation

This section contains a listing of MYO and COI documentation, as well as links to various Intel® Xeon Phi™ coprocessor collateral documents.

13.1 MYO documentation

MYO tutorials and other documents location on Windows*:  
C:\Program Files\Intel\MPSS\sdk\tutorials\myo

13.2 COI documentation

COI documentation for Windows*:  
(MPSS EXE Installation Base)\docs\coi\release_notes.txt  
(MPSS EXE Installation Base)\docs\coi\MIC_COI_API_Reference_Manual_1_0.pdf  
(INTEL_MPSS_HOST_SDK)\include\intel-coi\ - header files contain full API descriptions  
(INTEL_MPSS_HOST_SDK)\tutorials\coi\README_Windows_1.txt - instructions on how to get coprocessor side binaries and general building and use of tutorials

13.3 Intel® Xeon Phi™ coprocessor collateral

Intel® Xeon Phi™ coprocessor Product Brief:  

Intel® Xeon Phi™ coprocessor Specification Update:  

Intel® Xeon Phi™ coprocessor Safety and Compliance Guide:  

Intel® Xeon Phi™ coprocessor Datasheet:  

Intel® Xeon Phi™ coprocessor Software Users Guide:
Intel® Xeon Phi™ coprocessor System Software Developers Guide:

Intel® Xeon Phi™ coprocessor Developers Quick Start Guide:

Intel® Xeon Phi™ coprocessor System Administration Guide:

Intel® Xeon Phi™ coprocessor Instruction Set Architecture Reference Manual: