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 Phi, and Xeon 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>Version 3.8</td>
<td>Document update for the release of Intel® MPSS 3.8</td>
<td>October 2016</td>
</tr>
<tr>
<td>Version 3.7</td>
<td>Initial draft of the 3.7 Intel® MPSS User’s Guide</td>
<td>November 2015</td>
</tr>
<tr>
<td>Version 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  About this manual ............................................................................................................. 11  
   1.1 Overview of this document.............................................................................................. 11  
   1.2 Intel® MPSS release history............................................................................................ 12  
      1.2.1 Technology previews in this release ................................................................. 12  
   1.3 Notational conventions ................................................................................................. 12  
      1.3.1 Symbols within normal text.................................................................................. 12  
      1.3.2 Code conventions ................................................................................................. 13  
      1.3.3 Directory symbols.................................................................................................. 13  
      1.3.4 Command syntax................................................................................................. 14  
   1.4 Terminology ................................................................................................................ 15  
2  Intel® MPSS at a glance ..................................................................................................... 17  
   2.1 Intel® Xeon Phi™ coprocessor hardware and system architecture................................. 17  
   2.2 Programming models and the Intel® MPSS architecture .............................................. 19  
      2.2.1 Programming models .......................................................................................... 19  
      2.2.2 Intel® MPSS software architecture and components ........................................... 21  
      2.2.3 Coprocessor networking ...................................................................................... 24  
   2.3 Supported productivity tools ......................................................................................... 28  
   2.4 Related documentation ................................................................................................. 28  
      2.4.1 SCIF documentation ......................................................................................... 28  
      2.4.2 SCIF tutorials location ....................................................................................... 29  
      2.4.3 COI documentation ......................................................................................... 29  
      2.4.4 MYO documentation ......................................................................................... 29  
      2.4.5 Micperf documentation ...................................................................................... 29  
      2.4.6 Intel® Xeon Phi™ coprocessor collateral .............................................................. 29  
3  Intel® Xeon Phi™ coprocessor installation process ............................................................ 31  
   3.1 Hardware and software prerequisites ............................................................................ 31  
      3.1.1 Host system hardware ......................................................................................... 31  
      3.1.2 BIOS configuration ............................................................................................ 31  
      3.1.3 Supported host operating systems ..................................................................... 31  
      3.1.4 Host operating system configuration ................................................................... 32  
      3.1.5 Root access ........................................................................................................ 32  
      3.1.6 SSH access to the coprocessor .......................................................................... 32  
      3.1.7 Init scripts .......................................................................................................... 33  
      3.1.8 Network manager ............................................................................................... 34  
   3.2 Physical installation of coprocessors ............................................................................ 35  
      3.2.1 Workstation considerations .............................................................................. 36  
      3.2.2 Cluster considerations ....................................................................................... 37  
   3.3 Validating physical installation of coprocessors ........................................................... 39  
   3.4 Base installation of the Intel® MPSS ............................................................................ 40  
      3.4.1 Obtaining the Intel® MPSS distribution ............................................................. 40  
      3.4.2 Uninstalling previous Intel® MPSS installation .................................................. 40  
      3.4.3 Rebuilding Intel® MPSS host drivers .................................................................. 41  
      3.4.4 Installing Intel® MPSS ....................................................................................... 42  
      3.4.5 Updating the coprocessor’s flash and SMC firmware ......................................... 42  
      3.4.6 Initializing Intel® MPSS default configuration settings ....................................... 45  
      3.4.7 Starting Intel® MPSS .......................................................................................... 46  
      3.4.8 Validating Intel® MPSS installation .................................................................... 47
3.4.9 Running "Hello world" ........................................................................... 48
3.5 Summary of the installation process ............................................................ 50
3.6 Basic network configuration ........................................................................ 50
  3.6.1 MAC address assignment ..................................................................... 50
  3.6.2 IP address considerations for external bridging ................................. 51
  3.6.3 Configuring a basic external bridge .................................................... 51
  3.6.4 Defining and implementing exported/mounted file systems ............. 52
  3.6.5 Configuring the host firewall ............................................................ 53
  3.6.6 Installing Lustre* on the coprocessor ............................................... 54
3.7 Installing OFED ....................................................................................... 55
  3.7.1 Supported OFED distributions .......................................................... 56
  3.7.2 Installing TSR ................................................................................... 56
  3.7.3 Installing OFED-3.12-1 ..................................................................... 58
  3.7.4 Installing OFED 3.18-2 ..................................................................... 58
  3.7.5 Installing Mellanox* OFED 2.4 .......................................................... 59
  3.7.6 Starting OFED .................................................................................. 59
  3.7.7 Stopping/restarting OFED ............................................................... 60
  3.7.8 Validating OFED installation ............................................................ 60
4 Configuring and booting the coprocessor OS ................................................. 62
  4.1 Assisted configuration and control ............................................................ 63
    4.1.1 Configuration files .......................................................................... 63
    4.1.2 Initializing, updating and resetting the configuration files ............... 68
    4.1.3 Micctrl directory path modifiers .................................................... 69
    4.1.4 Boot configuration ......................................................................... 72
    4.1.5 Assisted boot process .................................................................... 74
  4.2 Manual configuration and control ............................................................. 76
    4.2.1 Directly editing (and persisting) coprocessor /etc files .................... 76
    4.2.2 NFS mounting the root and other file systems ............................... 78
    4.2.3 Driver sysfs settings ...................................................................... 78
    4.2.4 Coprocessor-side kernel commandline parameters ....................... 79
    4.2.5 Controlling the coprocessor ........................................................... 79
5 Networking configuration ............................................................................... 82
  5.1 Assisted configuration ............................................................................. 82
    5.1.1 Host SSH keys .............................................................................. 83
    5.1.2 Name resolution configuration ....................................................... 83
    5.1.3 Host name assignment ................................................................... 83
    5.1.4 MAC address assignment ............................................................. 83
  5.2 Network topologies ................................................................................. 84
    5.2.1 Static pair configuration .................................................................. 84
    5.2.2 Internal bridge configuration .......................................................... 86
    5.2.3 External bridge configuration ......................................................... 89
  5.3 Manual configuration ............................................................................... 91
    5.3.1 Host name ..................................................................................... 91
    5.3.2 Mac addresses ............................................................................... 91
    5.3.3 Network topologies ....................................................................... 92
  5.4 IPoIB networking configuration ............................................................... 96
    5.4.1 Managing the IPoIB interface .......................................................... 96
    5.4.2 IP addressing .................................................................................. 97
    5.4.3 Datagram vs. connected modes ...................................................... 97
6 User credentialing and authentication .......................................................... 98
6.1 Assisted configuration of user credentials ........................................... 98
  6.1.1 Local configuration ........................................................................ 98
  6.1.2 Enabling the LDAP service ............................................................ 100
  6.1.3 Enabling the NIS service .................................................................. 100
6.2 Manual configuration of user credentials .............................................. 101
  6.2.1 Configuration file based credentialing .............................................. 101
  6.2.2 Enabling SSH host based authentication ........................................ 104

7 Adding software to the coprocessor's file system .................................... 105
  7.1 Adding individual files to a host resident file system image ................... 105
    7.1.1 Assisted configuration .................................................................. 105
    7.1.2 Manual configuration .................................................................... 106
    7.1.3 Installing RPMs .......................................................................... 107
  7.2 Adding software to the coprocessor's file system ................................... 108
    7.2.1 Installing RPMs .......................................................................... 108
    7.2.2 Preserving the modified file system .............................................. 111

8 Compilation for the coprocessor ............................................................... 113
  8.1 Cross compiling software with the Intel® MPSS SDK ............................. 113
    8.1.1 SDK overview ............................................................................. 113
    8.1.2 Cross compilation of gnu build system based packages ................. 113
    8.1.3 Example case: zsh ..................................................................... 115
    8.1.4 Cross compiling with ICC .............................................................. 119
  8.2 Native compilation ............................................................................. 119
    8.2.1 Creating and adding a repo ............................................................ 120
    8.2.2 Installing the development tool chain .......................................... 120
    8.2.3 Configuring the build directory .................................................... 120
    8.2.4 Making and installing the package ................................................. 121

9 Intel® MPSS component configuration and tuning ..................................... 123
  9.1 The coprocessor OS configuration and tuning ....................................... 123
    9.1.1 Clock source for the coprocessor .................................................. 123
    9.1.2 Process oversubscription ............................................................... 123
    9.1.3 Verbose logging ........................................................................... 124
    9.1.4 Cgroup memory control ............................................................... 124
    9.1.5 Power management control .......................................................... 125
    9.1.6 VFS optimizations ...................................................................... 125
  9.2 Host driver configuration ..................................................................... 125
    9.2.1 Lost node watchdog ..................................................................... 125
    9.2.2 Watchdog auto-reboot .................................................................. 126
    9.2.3 Crash dump capture ..................................................................... 126
  9.3 SCIF configuration .............................................................................. 126
    9.3.1 Peer to peer (p2p) support ............................................................. 126
    9.3.2 Peer to peer proxy control ............................................................. 127
    9.3.3 Ulimit checks for max locked memory in scif ................................ 127
    9.3.4 Registration caching ..................................................................... 127
    9.3.5 Registration caching limit ............................................................. 128
    9.3.6 Huge page support ...................................................................... 128
  9.4 COI configuration .............................................................................. 128
    9.4.1 COI offload user options ............................................................... 128
  9.5 Virtual console configuration and access .............................................. 130
  9.6 Virtio block device configuration and use ........................................... 131
    9.6.1 Using a virtio block device as an ext2 file system ......................... 131
C.2 The mic.ko driver sysfs entries ................................................................. 183
C.2.1 Hardware information ............................................................................. 183
C.2.2 State entries ............................................................................................ 184
C.2.3 Statistics .................................................................................................. 185
C.2.4 Debug entries .......................................................................................... 185
C.2.5 Flash entries ........................................................................................... 186
C.2.6 Power management entries ..................................................................... 186
C.2.7 Other entries ........................................................................................... 186

D Micrasd ........................................................................................................... 188
E Micractiveloadex .......................................................................................... 189
F Optional Intel® MPSS components ................................................................. 190
  F.1 Intel® MPSS ganglia support ..................................................................... 190
     F.1.1 Requirements ....................................................................................... 190
     F.1.2 Installing ganglia on the host ................................................................. 191
     F.1.3 Installing intel® MPSS ganglia RPMs on the coprocessor ................. 193
     F.1.4 Starting Intel® MPSS with ganglia support ......................................... 193
     F.1.5 Stopping Intel® MPSS with ganglia support ........................................ 193
  F.2 Performance workloads (micperf) .............................................................. 193
     F.2.1 Installation requirements ...................................................................... 194
     F.2.2 Distributed files .................................................................................... 195
     F.2.3 RPM installation ................................................................................... 195
     F.2.4 Python* installation ............................................................................ 195
     F.2.5 Alternative to Python* installation ...................................................... 196
  F.3 Intel® MPSS reliability monitor support ..................................................... 196
     F.3.1 Requirements ....................................................................................... 196
     F.3.2 Install Intel® MPSS with reliability monitor support ......................... 196
     F.3.3 Starting Intel® MPSS with reliability monitor support ....................... 197
     F.3.4 Stopping Intel® MPSS with reliability monitor support ..................... 197
     F.3.5 Reliability monitor configuration file and log ...................................... 197

G Rebuilding Intel® MPSS components ............................................................... 199
  G.1 Recomping the Intel® MPSS ganglia modules ............................................ 199
  G.2 Recomping the Intel® MPSS MIC management modules ......................... 200
  G.3 How to extract and use the COI open source distribution ......................... 201
     G.3.1 Building COI libraries and binaries ................................................... 201
     G.3.2 Installing the host library .................................................................. 201
     G.3.3 Installing the coprocessor-side binaries and libraries ....................... 201
     G.3.4 Building and running the COI tutorial .............................................. 202
  G.4 How to extract and use the MYO open source distribution ..................... 203

H Services tutorial ............................................................................................. 204
  H.1 Service startup by priorities (RHEL* 6.x) .................................................. 204
  H.2 Service startup by dependencies (SLES* 11SP4) ....................................... 205
  H.3 Service start priority on the coprocessor ................................................. 206

I Troubleshooting and debugging ....................................................................... 207
  I.1 Log files ..................................................................................................... 207
     I.1.1 Dmesg output ...................................................................................... 207
     I.1.2 Syslog output ....................................................................................... 207
  I.2 Coprocessor post codes ............................................................................. 208
  I.3 Installing Intel® MPSS debug information ............................................... 211
I.4 Kernel crash dump support ............................................................... 211
I.5 Gnu Debugger (gdb) for the coprocessor ........................................... 212
I.5.1 Running gdb natively on the coprocessor ..................................... 212
I.5.2 Running remote gdb on the coprocessor .................................... 212
I.5.3 Gdb remote support for data race detection ................................. 213
I.5.4 Debugging heterogeneous/offload applications ............................ 213
I.5.5 Enabling mic gdb debugging for offload processes....................... 214
List of figures

Figure 1: Typical Intel® Xeon Phi™ Based Workstation Configuration .............................................17
Figure 2: Intel® Xeon Phi™ coprocessor-Based Compute Node within a Cluster .................................18
Figure 3: Intel® Xeon Phi™ Architecture Ring and Cores ....................................................................19
Figure 4: Spectrum of Programming Models .......................................................................................20
Figure 5: Intel® MPSS Components ......................................................................................................21
Figure 6: Static Pair Configuration .........................................................................................................25
Figure 7: Internal bridge network ..........................................................................................................26
Figure 8: External bridge network ..........................................................................................................27
Figure 9: Uniform distribution of Intel® Xeon Phi™ coprocessors ..........................................................36
Figure 10: Two Intel® Xeon Phi™ coprocessors Installed in the Same I/O Hub ....................................37
Figure 11: Intel® Xeon Phi™ and InfiniBand® HCA sharing an I/O hub ..................................................38
Figure 12: Symmetric Distribution of coprocessors and HCAs .............................................................39
Figure 13: One-to-One IB Device (HCA, Port) Mapping ........................................................................96

List of tables

Table 1: Productivity Tools Supported by Intel® MPSS 3.8 .................................................................28
Table 2: Supported Host Operating Systems .........................................................................................32
Table 3: System V format commands ....................................................................................................34
Table 4: File System Characteristics .....................................................................................................53
Table 5: OFED Distribution vs. Supported Features ............................................................................56
1 About this manual

This manual is intended to provide you with an understanding of the Intel® Manycore Platform Software Stack (Intel® MPSS), show what it is, demonstrate how to configure it, and explain how to use its components.

This section begins with an overview of the remainder of the document, presents notation used in this document, lists further documentation available for selected Intel® MPSS components, and concludes with a table of terminology.

It is recommended to review at least sections 1-3 prior to installing the software stack for the first time.

1.1 Overview of this document

Section 2 provides a high level overview of the coprocessor architecture and then gives an overview of the Intel® MPSS architecture.

Section 3 is a thorough, step-by-step guide to installing Intel® MPSS, including basic configuration steps and considerations for both workstation and cluster environments.

Section 4 is an in-depth discussion on configuring the coprocessor and the software stack.

Section 5 describes supported network configurations, indicates when each might be used, and instructs how to configure each of them. It also discusses how to configure NFS mounts, and DHCP.

Section 6 describes how to configure user credentials on the coprocessor.

Section 7 presents methods for adding software to the coprocessor's file system.

Section 8 explains how to cross-compile software for execution on the coprocessor, as well as how to compile and build on the coprocessor itself (native build).

Section 9 presents configuration options for Intel® MPSS components, including the coprocessor Linux* kernel, the host driver, the SCIF communication API, the COI offload interface, the virtual console, and the Virtio block device.

Appendix A describes all Intel® MPSS-specific configuration parameters.

Appendix B describes all Intel® MPSS micctrl commands.

Appendix C presents sysfs entries exposed by the Intel® MPSS host driver.

Appendix D provides some details on the micrasd daemon.

Appendix E describes the micnativeloadex utility.

Appendix F provides detailed instructions on installing several optional components: Ganglia, Micperf and Reliability Monitor.
Appendix G provides instructions for rebuilding selected Intel® MPSS components.

Appendix H is a tutorial describing how services are started on supported Linux* host Operating Systems and the coprocessor.

Appendix I presents several tools and techniques that can be used in troubleshooting and debugging.

1.2 Intel® MPSS release history

This version of the Intel® MPSS User’s Guide covers the release 3.8.

Beginning with the Intel® MPSS 3.2 release, the significant new features in each release are described in a document entitled Prominent features of the Intel® Manycore Platform Software Stack (Intel® MPSS) version M.N, where M.N is the software stack release number. These documents can be found by searching on https://software.intel.com/en-us/mic-developer.

1.2.1 Technology previews in this release

CCL-direct for kernel mode clients

This release includes a technology preview of CCL-Direct for kernel mode clients. This includes an experimental version of kernel mode InfiniBand* verbs and RDMA_CM, and an experimental version of IPoIB. This experimental version of CCL-Direct kernel mode support was tested with a Lustre* client. Refer to the /usr/share/doc/ofed-driver-*/*lustre-phi.txt document for information on how to build and install a Lustre* client on the coprocessor. This preview supports the Mellanox* mlx4 driver and associated hardware, and currently supports various versions of the OFED software. See Section 5.4 for information on IPoIB networking configuration.

File IO performance improvements

This Intel® MPSS technology preview is intended to improve the performance of system calls that read and write to files on tmpfs and ramfs mount points. In addition to a set of kernel configuration parameters that enable these optimizations (ON by default in the release), kernel command line options provide additional control to enable or disable the read and write optimizations.

See Section 9.1.6 for configuration instructions.

1.3 Notational conventions

1.3.1 Symbols within normal text

This guide Italicizes commands and their arguments when they appear in prose sections of the document. For example: micctrl now executes ifup micN for each of the coprocessors.

This guide also italicizes the software stack configuration parameter names when they appear in prose sections. For example: When the RootDevice parameter <type> is NFS or SplitNFS...
Files and directories in prose sections are italicized. For example: /etc/mpss/default.conf.

micN denotes any coprocessor name of the form mic0, mic1, etc. where N=0, 1, 2, ..., 255; this convention is typically used in file names. For example, the file name micN.conf denotes any of the mic0.conf, mic1.conf, ..., files.

Emboldened text indicates the exact characters you type as input. It is also used to highlight the elements of a graphical user interface such as buttons and menu names. For example: press the ENTER button, select Copy from the Edit menu.

1.3.2 Code conventions

There are code snippets throughout this document.

COURIER text denotes code and commands entered by the user.

Italic COURIER text denotes terminal output by the computer.

"[host]" at the beginning of a line denotes a command entered on the host with user or root privileges.

"[host]#" at the beginning of a line denotes a command entered on the host with root privileges.

"[micN]" at the beginning of a line denotes a command entered on a coprocessor with user or root privileges.

"[micN]#" at the beginning of a line denotes a command entered on a coprocessor with root privileges.

The example below shows the micctrl --config command executed as a non-root user, and the truncated output it generated:

[host]$ micctrl --config
mic0:
=================================================================
Config Version: 1.1

Linux Kernel: /usr/share/mpss/boot/bzImage-knightscorner:

1.3.3 Directory symbols

For convenience, we define several symbols that denote commonly referenced directories.

$MPSS38 is the top directory into which the mpss-3.8-linux.tar file has been extracted.

$MPSS38_K1OM is the directory into which the mpss-3.8-k1om.tar file has been extracted. Usually it is $MPSS38/k1om.

$MPSS38_SRC is the directory into which the mpss-src-3.8.tar file has been extracted. Usually it is $MPSS38/src.
$DESTDIR$ is a symbol that indicates the directory path variable that micctrl prepends to all micctrl accesses of micctrl created files. Refer to Appendix B.2.1 for details.

$CONFIGDIR$ is a symbol that indicates the directory path variable at which micctrl creates software stack-specific configuration files. Refer to Appendix B.2.1 for details.

$VARDIR$ is a symbol that indicates the directory path variable at which the micctrl --initdefaults and --resetconfig commands create the common and micN overlay hierarchies, and at which the micctrl --rootdev command places a ramfs file system image or NFS file system hierarchy. Refer to Appendix B.3.2.1 for details.

$SRCDIR$ is a symbol that indicates the directory path at which the micctrl --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands look for the coprocessor’s Linux* kernel image and default file system image. Refer to Appendix B.3.2.2 for details.

$NETDIR$ is a symbol that indicates the directory path at which the micctrl --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands create and/or edit control files. Refer to Appendix B.3.2.3 for details.

### 1.3.4 Command syntax

Shown below are conventions used in micctrl command syntax and Intel® MPSS configuration parameter syntax:

<...> indicates a variable value to be supplied.

[...] indicates an optional component.

(x|y)...(z) is used in micctrl command syntax and the software stack configuration parameter syntax to indicate a choice of values.

The syntax of the Overlay configuration parameter is:

```
Overlay (Filelist|Simple|File) <source> <target> (on|off)
Overlay RPM <source> (on|off))
```

It indicates that there are two basic forms. The first takes a Filelist or Simple or File type, followed by <source> and <target> values to be provided, followed by a choice of on or off. The second form takes the RPM type, followed by only a <source> value to be provided, followed by a choice of on or off.

The syntax of the micctrl --userupdate command:

```
micctrl --userupdate=(none|overlay|merge|nochange) \[-a|--pass=](none|shadow) [--nocreate]
```

It indicates that the userupdate method must be set to one of none, overlay, merge, or nochange. An optional argument can be invoked using either -a or --pass and must specify one of none or shadow (For Example: -a none or --pass=none). Finally, there is an optional --nocreate command.

The --nocreate option is italicized, indicating that it is a common suboption, which can be issued with multiple commands. For brevity, common suboptions are defined once in Appendix B.3.2.
## 1.4 Terminology

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI</td>
<td>Application binary interface</td>
</tr>
<tr>
<td>CCL</td>
<td>Coprocessor Communication Link</td>
</tr>
<tr>
<td>COI</td>
<td>Coprocessor Offload Infrastructure</td>
</tr>
<tr>
<td>Coprocessor</td>
<td>Intel® Xeon Phi™ coprocessor</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>Ganglia</td>
<td>A distributed monitoring system</td>
</tr>
<tr>
<td>GDB</td>
<td>Gnu debugger</td>
</tr>
<tr>
<td>HCA</td>
<td>Host Channel Adapter</td>
</tr>
<tr>
<td>IPoIB</td>
<td>Internet Protocol over InfiniBand*</td>
</tr>
<tr>
<td>K1OM</td>
<td>Architecture of the Intel® Xeon Phi™ coprocessor</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>Lustre</td>
<td>A parallel, distributed file system</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MIC</td>
<td>Many Integrated Cores, an informal name for the KNC architecture</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>MPSS</td>
<td>Intel® Manycore Platform Software stack</td>
</tr>
<tr>
<td>MYO</td>
<td>Mine, Yours, Ours shared memory infrastructure</td>
</tr>
<tr>
<td>NIS</td>
<td>Network Information System</td>
</tr>
<tr>
<td>OFED</td>
<td>Open Fabric Enterprise Distribution</td>
</tr>
<tr>
<td>PCIe</td>
<td>PCI Express</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PCIe2</td>
<td>PCI Express 2.0</td>
</tr>
<tr>
<td>QPI</td>
<td>Intel® QuickPath Interconnect, a point-to-point processor interconnect</td>
</tr>
<tr>
<td>RHEL</td>
<td>Red Hat® Enterprise Linux®</td>
</tr>
<tr>
<td>RPM</td>
<td>RPM package manager</td>
</tr>
<tr>
<td>SCIF</td>
<td>Symmetric Communication Interface</td>
</tr>
<tr>
<td>SLES</td>
<td>SUSE® Linux® Enterprise Server</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric Multi-Processor</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Drive</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>Sysfs</td>
<td>A virtual file system</td>
</tr>
<tr>
<td>VEth</td>
<td>Virtual Ethernet</td>
</tr>
</tbody>
</table>

§
This section provides an overview of Intel® MPSS. It begins with a high level description of Intel® Xeon Phi™ coprocessor hardware and system architecture. The section also discusses programming models that the software stack is designed to support, shows how its various components support those programming models, and provides a description of supported network configurations. It concludes with a listing of other available documentation.

2.1 Intel® Xeon Phi™ coprocessor hardware and system architecture

The coprocessor is a PCIe add-in card that has been designed to be installed into an Intel® Xeon®-based platform. A typical platform configuration consists of one or two processors and one or two coprocessors. A typical configuration is shown in Figure 1.

![Figure 1: Typical Intel® Xeon Phi™ Based Workstation Configuration](image)

When one or more PCIe based InfiniBand* host channel adapters, such as Intel® True Scale HCA, are installed in the platform, coprocessors may communicate at high speed with Intel® Xeon® processors and coprocessors in other platforms in a cluster configuration. Figure 2 shows a typical coprocessor-based compute node within a cluster. In a single system, the coprocessors can communicate with each other through the PCIe peer-to-peer interconnect without any intervention from the host. Other configurations are discussed in Section 3.2.
The coprocessor is composed of up to 61 processor cores, caches, memory controllers, PCIe client logic, and a very high bandwidth, bidirectional ring interconnect (Figure 3). Each of the cores comes complete with a private L2 cache that is kept fully coherent by a global-distributed tag directory. The coprocessors’ K1OM architecture cores support an x86 instruction set with additional vector instructions unique to the architecture. The coprocessors’ K1OM ABI differs from the Intel® Xeon® processor’s ABI. For these reasons, Intel® Xeon® processor binaries cannot be run on the coprocessor, and vice versa.

The memory controllers and the PCIe client logic provide a direct interface to the GDDR5 memory on the coprocessor and the PCIe bus, respectively. All these components are joined by the ring interconnect.

The coprocessors do not have a permanent file system storage, such as an SSD. Instead the file system is maintained in RAM and/or is remotely (for instance: NFS) mounted.

Each coprocessor runs a standard Linux® kernel (2.6.38 as of this writing) with some minor accommodations for the MIC hardware architecture. Because it runs its own OS, the coprocessor is not hardware cache coherent with the host’s Intel® Xeon® processors or other PCIe devices.
Figure 3: Intel® Xeon Phi™ Architecture Ring and Cores

For more information on Intel® Xeon Phi™ coprocessor architecture, visit the Intel® Xeon Phi™ Product Family page.

2.2 Programming models and the Intel® MPSS architecture

To understand the software stack architecture, it is useful to grasp the range of programming models supported by Intel® MPSS and the coprocessor.

2.2.1 Programming models

The Offload, Symmetric and Native (MIC-hosted) programming models offer a diverse range of usage models. An overview of these options are depicted in Figure 4.
2.2.1.1 Offload programming model

In the Offload model, one or more processes of an application are launched on one or more Intel® Xeon® host processors. These processes, represented in the figure by `main()`, can offload computation, represented by `work()`, to one or more attached coprocessors to take advantage of their many-core architecture, wide vector units and high memory bandwidth. In case the application is composed of more than one process, the processes often communicate using some form of message passing, such as Message Passing Interface (MPI) thus we also show `MPI_*()`, on the host. This offload process is programmed via the use of offload pragmas supported by the Intel® C/C++ and FORTRAN compilers. When an application is created with one of these compilers, offloaded execution will fall back to the host in the event that a coprocessor is not available. This is why an instance of `work()` is also shown on the host.

2.2.1.2 Symmetric programming model

The Symmetric programming model is convenient for an existing HPC application which is composed of multiple processes, each of which could run on the host or coprocessor, and use some standard communication mechanism such as MPI. In this model, computation is not offloaded, but rather remains within each of the processes comprising the application. In such cases, where the application is MPI based, the OFED distributions enable high bandwidth/low latency communication using installed Intel® True Scale or Mellanox* InfiniBand* Host Communication Adapters.

2.2.1.3 Native programming model

The Native (MIC-hosted) programming model is a variant of the Symmetric model in which one or more processes of an application are launched exclusively on the coprocessors. From the software stack architecture perspective, these programming
models typically depend on SCIF and the VEth (Virtual Ethernet) driver to launch processes on the coprocessors.

### 2.2.2 Intel® MPSS software architecture and components

Figure 5 provides a high level representation of Intel® MPSS and its relation to other important software components. The host software stack is shown to the left and the coprocessor software stack to the right. It is worth noting that although the stacks are mostly symmetric, their components (including applications) are not binary compatible.
2.2.2.1 The coprocessor operating system

Underlying all computation on the coprocessor is a standard Linux* kernel (2.6.38 as of this writing) with some minor accommodations for the MIC architecture, such as for saving the state of the extended MIC register set on a context switch. The Linux* kernel and initial file system image for the coprocessor are installed into the host file system as part of the Intel® MPSS. After installation, the coprocessor’s Linux* needs to be configured according to the expected workload/application. The coprocessor OS configuration will be covered in detail starting in Section 4.

The Linux* environment on the coprocessor utilizes BusyBox* to provide a number of Linux* utilities. These utilities provide the basic functionalities, and may seem limited when compared with the host Linux* distribution. For more information regarding BusyBox*, see the link [http://www.busybox.net/](http://www.busybox.net/).

2.2.2.2 Intel® MPSS middleware libraries

The compiler runtimes depend on the Coprocessor Offload Infrastructure (COI) library to offload executables and data for execution on the coprocessor, and use Mine Your Ours (MYO) shared memory infrastructure to provide a virtual shared memory model that simplifies data sharing between processes on the host and each coprocessor. Similarly, some functions in the Intel® Math Kernel Library (MKL) automatically offload workloads using the COI library.

COI, MYO, and other Intel® MPSS components rely on the Symmetric Communication Interface (SCIF) user mode API for PCIe communication services between the host processor, coprocessors, and InfiniBand® host channel adapters. SCIF delivers very high bandwidth data transfers and sub-μsec write latency to memory shared across PCIe, while abstracting the details of communication over PCIe.

The COI, MYO, and SCIF libraries are also available for use by other applications. Section 2.4 lists additional documentation on these libraries.

2.2.2.3 Intel® MPSS modules and daemons

The host driver (mic.ko) is the component of Intel® MPSS that initializes, boots, and manages each coprocessor. To boot a coprocessor, mic.ko injects the Linux* kernel image and a kernel command line into the coprocessor’s memory and signals it to begin execution. The host driver also includes a virtual console driver. Finally, mic.ko directs power management of the installed coprocessors.

Virtual Ethernet (VEth) drivers on the host and coprocessor implement a virtual Ethernet transport between them. Support is provided for standard TCP/UDP/IP stack and tools, such as ssh, scp, etc., across PCIe.

The virtio block device (virtblk) uses the Linux* virtio data transfer mechanism to implement a block device on the coprocessor. The device stores data on a specified storage location on the host and can therefore be persistent across coprocessor’s reboots.

SCIF functionality is largely implemented in kernel mode SCIF drivers on the host and the coprocessor.
The Intel® True Scale and Mellanox® drivers enable direct data transfers between each coprocessor’s memory and an installed Intel® True Scale or Mellanox® InfiniBand® HCA. Intel® MPSS also includes an optional InfiniBand® over SCIF (ibscif) driver which emulates an InfiniBand® HCA to the higher levels of the OFED stack. This driver uses SCIF to provide high bandwidth, low latency communication between multiple coprocessors in a host platform, for example between MPI ranks on separate coprocessors.

An mpssd daemon runs on the host, and directs the initialization and booting of each coprocessor based on a set of configuration files. The daemon is started and stopped with the Linux® mpss service, and instructs the coprocessors to boot or shutdown. In the event that the coprocessor’s OS crashes, mpssd will reboot it or bring it to a ready (to be booted) state. A micmpssd daemon on the coprocessor communicates with mpssd to perform operations, such as dynamically modifying user credentials, on behalf of micctrl.

micrasd is an application that runs on the host to handle and log hardware errors reported by coprocessors. It is normally controlled through the micras service. Refer to Appendix C.2.7 for additional information.

### 2.2.2.4 Tools and utilities

Intel® MPSS includes several system management tools and utilities:

*micctrl* is a utility which enables users to control (boot, shutdown, reset) each (or all) of the installed coprocessors. The utility also offers numerous options to simplify the configuration process. Configuration tasks can include controlling user access to coprocessors, adding them to a TCP/IP network, and installing software into the software stack-supplied default coprocessor file system (the default initramfs).

A substantial portion of this document is devoted to creating an optimized configuration, which can be accomplished either by using *micctrl* or by directly editing configuration files (or both). *micctrl* is discussed at length throughout this document. Its commands are described in detail in Appendix B. The same information is available online from *micctrl* help:

```
[host]$ micctrl -h
```

*micinfo* and *mpssinfo* display information about the coprocessors installed in the system as well as information about the host’s OS and Intel® MPSS host driver. *mpssinfo* is a POSIX compliant version of *micinfo*. For detailed information, refer to the *micinfo* or *mpssinfo* man page.

```
[host]$ man micinfo
[host]$ man mpssinfo
```

*micflash* and *mpssflash* are used to update a coprocessor’s flash image, save it to a file on the host, and display the currently loaded flash version. *mpssflash* is a POSIX-compliant version of *micflash*. For detailed information about *micflash* or *mpssflash*, refer to their man pages:

```
[host]$ man micflash
[host]$ man mpssflash
```
The *micsmc* tool is used to monitor coprocessor statistics such as core utilization, temperature, memory usage, power usage statistics, and error logs. *micsmc* can function in two modes: GUI mode and command-line (CLI) mode. 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. For detailed information about *micsmc*, refer to its man page:

```
[host]$ man micsmc
```

The *miccheck* utility executes a suite of diagnostic tests that verify the configuration and current status of the coprocessor software stack. For detailed information about *miccheck*, refer to its man page:

```
[host]$ man miccheck
```

The *micnativeloadex* utility copies a coprocessor native binary to a specified coprocessor and executes it. Refer to Appendix E for additional information.

### 2.2.2.5 Optional packages

The software stack distribution includes several optional packages that may be installed. Additional information and installation instructions can be found in Appendix F:

- **Reliability Monitor**, is an optional service that runs on the head node of a cluster, and monitors health of MIC based compute nodes.

- The coprocessor Performance Workloads package can be used to evaluate the performance of an Intel® Xeon Phi™ coprocessor-based installation.

- An optional Ganglia support package enables Ganglia-based monitoring of the coprocessors (Ganglia is an open source cluster monitoring system).

### 2.2.6 Gcc toolchain

The Intel® MPSS distribution includes both a cross-compile gcc toolchain and a native gcc toolchain. The cross-compile toolchain is used from the host to build components for execution on coprocessors. Similarly, the native toolchain has the same task but runs on a coprocessor. The native gcc toolchain is not installed into the default coprocessor file system image, but is available in a separate tarball, which contains hundreds of binary RPMs that can be used to customize the default file system image. Among those packages are system daemons including *cron, rpcbind*, and *xinetd*; performance and debugging tools including *gprof, lsof, perf*, and *strace*; utilities including *bzip2, curl, rsync*, and *tar*; scripting languages including *awk, perl*, and *python*; and development tools including *autotools, bison, cmake, flex, git, make, patch*, and *subversion*.

### 2.2.3 Coprocessor networking

There are three basic network configuration options that enable the coprocessors to operate in a wide range of networking environments. These are briefly described below. Network configuration is described in depth in Section 5.
2.2.3.1 Static pair configuration

The static pair configuration creates a separate private network between the host and each coprocessor. It assigns an IP address to each of the network endpoints. Various options for selecting the IP addresses (as seen by the host) and the host’s IP address (as seen by the coprocessor) are available.

Figure 6 depicts a host, on the left, with two coprocessors. A private network was configured between the host and each coprocessor. Notice that mic0 and mic1 are on separate subnets.

![Figure 6: Static Pair Configuration](image)

This network configuration is established by default when the `micctrl --initdefaults` command is called for the first time. This configuration is sufficient for Intel® C++ and FORTRAN compiler pragma-based offload computation on a standalone (non-clustered) host platform and other program models where a coprocessor only requires a network connection to the host.

Section 5 provides additional information about this network configuration.

2.2.3.2 Bridged network configurations

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.

2.2.3.2.1 Internal bridge configuration

Some distributed applications running on coprocessors on a single node need to communicate between themselves, and, perhaps, with the host. An internal bridge
allows for the connection of one or more coprocessors within a single host system as a subnetwork. In this configuration, each coprocessor can communicate with the host and with other coprocessors in the platform. Figure 7 shows an example of an internal bridge configuration.

**Figure 7: Internal bridge network**

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

The additional considerations and steps to configure this network topology are described in Section 5.

**2.2.3.2.2 External bridge configuration**

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

**Figure 8** depicts a cluster in which the coprocessors on each host node are bridged to an external network. The IP addresses in such a 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 InfiniBand* networking for high bandwidth communication as application data transfer.
To prepare for configuration of 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.

These topics and steps to configure this network topology are described in Section 5.

Figure 8: External bridge network
2.3 Supported productivity tools

The following table lists compatible versions of productivity tools that are supported by Intel® MPSS release 3.8.

<table>
<thead>
<tr>
<th>Name of Tool</th>
<th>Supported Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Parallel Studio XE</td>
<td>2015, 2016</td>
</tr>
<tr>
<td>Intel® C++ Compiler</td>
<td>15.0, 16.0</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives for Linux*</td>
<td>8.2, 9.0</td>
</tr>
<tr>
<td>Intel® Math Kernel Library for Linux*</td>
<td>11.2, 11.3</td>
</tr>
<tr>
<td>Intel® Threading Building Blocks for Linux*</td>
<td>4.3, 4.4</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier XE</td>
<td>2015, 2016</td>
</tr>
<tr>
<td>Intel® SEP</td>
<td>3.11</td>
</tr>
<tr>
<td>GNU Compiler Collection (gcc)</td>
<td>4.8.3 20140911</td>
</tr>
<tr>
<td>GNU Binutils</td>
<td>2.23.52.0.1</td>
</tr>
</tbody>
</table>

**Note:** The `intel-composerxe-compat-k1om` RPM temporarily provides backward compatibility to ICC compiler versions prior to 14.0.0 via the soft links to `/opt/mpss/3.8/sysroot`. It is not a separate set of binaries for the x86_64-k1om-linux architecture used in Intel® MPSS 2.1.6720.

2.4 Related documentation

The Intel® Xeon Phi™ Coprocessor Developer Zone website has a wealth of information on all aspects of Intel® Xeon Phi™ coprocessor programming.

The following documentation specific to Intel® MPSS and Intel® Xeon Phi™ coprocessors is listed below.

2.4.1 SCIF documentation

The following SCIF documentation is installed during base Intel® MPSS installation:

```
$MPSS38/docs/SCIF_UserGuide.pdf  -SCIF User Guide
/usr/share/man/man3/scif*        -Man pages for SCIF API user mode
/usr/share/man/man9/scif*        -Man pages for SCIF API kernel mode
```
2.4.2 SCIF tutorials location

-SCIF tutorial source files
-Instructions for building and running the SCIF tutorials
-SCIF tutorial source package
-SCIF tutorial binaries package
-Debuggable SCIF tutorial binaries package

2.4.3 COI documentation

The following COI documentation is installed during base Intel® MPSS installation:

- release_notes.txt
- MIC_COI_API_Reference_Manual_1_0.pdf
- header files containing full API descriptions
- Full tutorials source and Makefiles
- man pages

2.4.4 MYO documentation

The following MYO documentation is installed during base Intel® MPSS installation:

- MYO tutorials and other documents
- header files containing full API descriptions
- man pages

2.4.5 Micperf documentation

When installed, Intel® Xeon Phi™ coprocessor Performance Workload (micperf) documentation can be found at /usr/share/doc/micperf-3.8 (see Appendix F.2).

2.4.6 Intel® Xeon Phi™ coprocessor collateral

The following documents provide additional information on various aspects of Intel® Xeon Phi™ product family hardware and software.

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 Instruction Set Architecture Reference Manual:

Information on platforms that support the Intel® Xeon Phi™ Coprocessor.

Intel® MPSS Performance Guide
3 Intel® Xeon Phi™ coprocessor installation process

This section describes the steps for installing and configuring Intel® Xeon Phi™ coprocessor hardware and software. Most of the steps through Section 3.4 are common to both workstation and cluster configurations. Section 3.6 and later are primarily of interest to cluster administrators and those with advanced workstation programming configuration requirements.

Note: It is strongly recommended that you read through this section before actually proceeding with installation, to ensure that all required components and facilities are available. It is also strongly recommended that these installation steps be performed in the order in which they are presented.

3.1 Hardware and software prerequisites

3.1.1 Host system hardware

A system that supports the coprocessor is required to run Intel® MPSS. You can find information on such platforms at the Intel® Developer Zone: https://software.intel.com/en-us/mic-developer. Search for an article entitled Which systems support the Intel® Xeon Phi™ coprocessor?

3.1.2 BIOS configuration

Several BIOS settings are important for the proper functioning of the software stack.

1. Enable large Base Address Registers (BAR)

BIOS and OS support for large (8GB+) Memory Mapped I/O Base Address Registers (MMIO BAR’s) above the 4GB address limit must be enabled.

In some instances, motherboard BIOS implementations have this feature set to disabled and it must be enabled manually.

2. Enable Intel® Turbo boost on the host platform

For best performance, it is recommended to enable Intel® Turbo Boost. Enabling this setting in platform’s BIOS is vendor specific.

3.1.3 Supported host operating systems

Intel® MPSS 3.8 has been validated against specific versions of Red Hat* Enterprise Linux* (RHEL*) and SUSE* Linux* Enterprise Server (SLES*) as the host operating system. Table 2 lists the supported versions of these operating systems.

To obtain the version of the kernel running on the host, execute:

```
[host]$ uname -r
```
Table 2: Supported Host Operating Systems

<table>
<thead>
<tr>
<th>Supported Host OS Versions</th>
<th>Kernel Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat* Enterprise Linux* 64-bit 6.7</td>
<td>2.6.32-573</td>
</tr>
<tr>
<td>Red Hat* Enterprise Linux* 64-bit 6.8</td>
<td>2.6.32-642</td>
</tr>
<tr>
<td>Red Hat* Enterprise Linux* 64-bit 7.1</td>
<td>3.10.0-229</td>
</tr>
<tr>
<td>Red Hat* Enterprise Linux* 64-bit 7.2</td>
<td>3.10.0-327</td>
</tr>
<tr>
<td>Red Hat* Enterprise Linux* 64-bit 7.3</td>
<td>3.10.0-500</td>
</tr>
<tr>
<td>SUSE* Linux* Enterprise Server 11 SP4 64-bit</td>
<td>3.0.101-63-default</td>
</tr>
<tr>
<td>SUSE* Linux* Enterprise Server 12 64-bit</td>
<td>3.12.28-4-default</td>
</tr>
<tr>
<td>SUSE* Linux* Enterprise Server 12 SP1 64-bit</td>
<td>3.12.49-11-default</td>
</tr>
<tr>
<td>SUSE* Linux* Enterprise Server 12 SP2 64-bit</td>
<td>4.4.16-56-default</td>
</tr>
</tbody>
</table>

Section 3.4.3 discusses rebuilding Intel® MPSS host drivers in the event that the host kernel has been patched/upgraded.

3.1.4 Host operating system configuration

1. The SUSE* Linux* Enterprise Server release kernel must be configured to allow non-SUSE* driver modules to be loaded. Edit the /etc/modprobe.d/unsupported-modules file and set the value of allow_unsupported_modules to 1.

2. If SELinux* is installed, it must be disabled before installing Intel® MPSS software to prevent SELinux* from overriding standard Linux* permissions settings.

3.1.5 Root access

Many tasks described in this document require root access privileges. Verify that you have such privileges to the machines you will configure.

Using sudo to acquire root privileges may result in subtle and undesirable side effects and should be done carefully. The sudo command might not retain the non-root environment of the caller. This could, for example, lead to using a different PATH environment variable than expected, resulting in execution of the wrong code.

When su is used to become root, the non-root environment is mostly retained. HOME, SHELL, USER, and LOGNAME are reset unless the -m switch is given. See the su man page for details.

3.1.6 SSH access to the coprocessor

Secure Shell (SSH) is a connectivity tool for secure remote command line login, command execution, and other services between two networked computers. SSH enables users to move native applications and data to the coprocessors and move the results back. Developers can use SSH to perform native compilation and other software development tasks.
Although most coprocessor configuration tasks can be done indirectly from the host, some administrators may prefer to use SSH to log on to a coprocessor to perform such configuration tasks directly or verify that a coprocessor's configuration is correct.

SSH access is generally not needed by users who will develop and/or execute offload applications using the Intel® C++ and FORTRAN offload pragmas.

The coprocessor's Linux* OS supports network access authentication using SSH keys or passwords; this requires that valid credentials exist on the coprocessor. Depending on parameterization, the micctrl --initidefaults command, when run after the base software stack installation (Section 3.4.4), creates root account and user accounts in each coprocessor's file system. The accounts created are based on the contents of the host's /etc/passwd file. In addition, if SSH key files are found in users' $HOME/.ssh directories, they will be propagated to each coprocessor's file system. This allows SSH access to any coprocessor without the need to enter a password.

It is recommended to install the latest version of the OpenSSH software on the coprocessor. The mpss-3.8-k1om.tar archive contains, among others, openssh*.k1om.rpm and open-ssh-without-openssl*.k1om.rpm packages that can be installed in the coprocessor's file system. Refer to Section 7 for instructions.

Each user, including root, that will need SSH access should generate a set of SSH keys by executing the ssh-keygen command prior to software stack installation:

```
[host]$ ssh-keygen
```

**Note:** It is recommended to use default key names and not to provide key passphrases.

See Section 6 for information on customizing the user credentialing behavior.

### 3.1.7 Init scripts

Red Hat* Enterprise Linux* 6 and SUSE* Linux* Enterprise Server 11 use the System V init system, while Red Hat* Enterprise Linux* 7 and SUSE* 12 utilize the systemd init system. System V makes use of the service command, which has the form:

```
service SCRIPT COMMAND [OPTIONS]
```

where the SCRIPT parameter specifies a System V init script, and the supported values of COMMAND depend on the invoked script.

Systemd uses the systemctl command, which has the form:

```
systemctl [OPTIONS...] COMMAND [NAME...]
```

where [NAME...] is zero or more parameters to the COMMAND.

The systemctl command is also used in RHEL* 7.x and SUSE* 12 instead of the chkconfig command.

Init commands in this document are in System V format. On host systems with RHEL* 7.x or SUSE* 12, those commands should be converted to systemd format as follows:
Table 3: System V format commands

<table>
<thead>
<tr>
<th>RHEL* 6/SUSE* 11 Command</th>
<th>RHEL* 7/SUSE* 12 Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>service mpss unload</td>
<td>systemctl stop mpss</td>
</tr>
<tr>
<td></td>
<td>modprobe -r mic</td>
</tr>
<tr>
<td>service SCRIPT COMMAND</td>
<td>systemctl COMMAND SCRIPT</td>
</tr>
<tr>
<td>chkconfig NAME on</td>
<td>systemctl enable NAME</td>
</tr>
<tr>
<td>chkconfig NAME off</td>
<td>systemctl disable NAME</td>
</tr>
</tbody>
</table>

For example, the command:

```
[host]# service nfs restart
```

should be converted to:

```
[host]# systemctl restart nfs
```

Similarly

```
[host]# chkconfig mpss on
```

becomes:

```
[host]# systemctl enable mpss
```

In the remainder of this document, `service` and `systemctl` are prepended with a superscript that links back to this section:

```
[host]# `service mpss start
```

### 3.1.8 Network manager

Some configuration of the network manager is required. Configuration process depends on the host operating system version.

**RHEL* 6, RHEL* 7 and SLES* 11:**

Users have encountered issues in configuring the virtual network interfaces for the coprocessors when NetworkManager was used on RHEL* 6, RHEL* 7 and SLES* 11 SP4 platforms. It is strongly recommended to use the older and more server-oriented network daemon instead with these operating systems.

To determine if NetworkManager is active, execute:

```
[host]# `service NetworkManager status
```
To switch to network daemon, perform the following on the host:

```
[host]# chkconfig NetworkManager off
[host]# chkconfig network on
[host]# service NetworkManager stop
[host]# service network start
```

**SLES* 12:**

The *wicked network configuration framework* should be enabled by default on SLES* 12. To verify whether it is running execute:

```
[host]$ systemctl status network.service
```

If the output shows another service force *wicked* by executing:

```
[host]# systemctl stop network.service
[host]# systemctl enable --force wicked.service
```

For proper functioning of the coprocessors’ networking with *wicked*, the *nanny* daemon should also be enabled. The recommended procedure is to create and/or edit the `/etc/wicked/local.xml` file to include a line that enables the *nanny* daemon, for example:

```
<config>
  <use-nanny>true</use-nanny>
</config>
```


After any change in network configuration, the *wicked* daemon should be restarted for configuration changes to take effect:

```
[host]# systemctl restart wicked
```

It is also recommended to flush the DNS cache by issuing:

```
[host]# systemctl restart nscd
```

### 3.2 Physical installation of coprocessors

*Note:* You can skip this section if your host has one CPU socket.

When installing coprocessors into a host platform, some consideration should be given to the slot or slots where they will be placed. The options depend on whether a coprocessor is to communicate with another PCIe devices, such as another coprocessor or an InfiniBand* HCA (such communication between PCIe devices is referred to as peer-to-peer - P2P).

Users should keep in mind that whenever PCIe devices are plugged into I/O hubs of different CPU sockets, communication between those devices will run across the Quick Path interconnect. The bandwidth of this communication will typically be lower than communication bandwidth between the two devices plugged into the same I/O hub. Therefore, if you expect to maximize P2P data transfers between two PCIe devices, it is
recommended to setup nodes with the above considerations in mind. Contact your host platform OEM or refer to motherboard documentation for information on bus/processor locality.

### 3.2.1 Workstation considerations

Workstations typically do not have InfiniBand® HCAs. In this case, P2P communication between coprocessors will determine how they are installed, and this is somewhat determined by the programming models, which were discussed briefly in Section 2.2. If there is just a single coprocessor, it makes little difference into which slot it is installed.

#### 3.2.1.1 Offload programming model

Most workstation applications use the offload programming model support provided by the Intel® C/C++ and FORTRAN compilers, and MKL libraries to offload work to one or more coprocessors. In this framework, the coprocessors communicate only with the host processor, not with each other, P2P bandwidth is thus not important. Therefore it is recommended that coprocessors are installed as uniformly as possible among the I/O hubs. Figure 9 is an example of such installation. Care, should be taken to ensure that the host side of the offload program is running on the same NUMA node as the coprocessor to which it is offloading work. Refer to the Intel® MPSS Performance Guide for additional information.

![Diagram of uniform distribution of Intel® Xeon Phi™ coprocessors](image)

**Figure 9: Uniform distribution of Intel® Xeon Phi™ coprocessors**

#### 3.2.1.2 Symmetric and native programming models

When an application is distributed across the coprocessors it is important to maximize the bandwidth between those devices. This might be the case, for example, when MPI ranks are instantiated on multiple coprocessors in a workstation. In this case, the best
performance might be achieved by installing pairs of coprocessors into each I/O hub as shown in Figure 10. Experimentation to find the best configuration is recommended.

Figure 10: Two Intel® Xeon Phi™ coprocessors Installed in the Same IO Hub

3.2.2 Cluster considerations

When both a coprocessor and an InfiniBand® HCA are installed in the same platform, it is important to maximize communication bandwidth between them. This suggests that, if there is one of each, they should be installed into the same I/O hub, as the example in Figure 11 shows.
By extension, when there are equal numbers of coprocessors and HCAs, we suggest installing the coprocessor and HCA pair into an I/O hub, as the example in Figure 12 shows. For other ratios of coprocessors and HCAs, consideration should be given to how the devices are expected to inter-communicate, remembering the relative communication BWs between the various PCIe slots in a platform.
3.3 Validating physical installation of coprocessors

Before installing and using Intel® MPSS, it is recommended to check if the host OS is able to enumerate and assign MMIO resources to the coprocessors. The `lspci` command, commonly found on Linux* installations, can be used to achieve this. The following shows typical output indicating the presence of a single coprocessor:

```
[host]$ lspci | grep -i Co-processor
08:00.0 Co-processor: Intel Corporation Device 225c (rev 20)
```

To verify that the BIOS/OS is able to assign all the required resources to the coprocessor, execute the following, noting that the earlier command reported that the coprocessor is on `bus:slot 08:00`.

```
[host]# lspci -s 08:00.0 -vv
08:00.0 Co-processor: Intel Corporation Device 225c (rev 20)
Subsystem: Intel Corporation Device 2500
Physical Slot: 4
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV-
VGA=Snoop- ParErr+ Stepping- SERR+ FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr-
DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 56
Region 0: Memory at 3c7e00000000 (64-bit, prefetchable)
[size=2000000000]
Region 4: Memory at ec000000 (64-bit, non-prefetchable)
[size=128K]
```

<output truncated>
The output shows that both BAR0 (region 0) and BAR1 (region 4) have valid assigned values.

If the expected number of coprocessors is not reported, it may help to reseat them before continuing. If the coprocessors are detected, but no resources have been assigned, check the system BIOS for support of large BARs (see Section 3.1.4).

3.4 Base installation of the Intel® MPSS

Base software stack includes all components that are needed to configure the Intel® MPSS environment, boot the installed coprocessors, and execute applications that use the offload or native execution models. Optional stack components must be installed if application processes running on the coprocessors are to communicate with supported InfiniBand* HCAs (see Section 3.7).

3.4.1 Obtaining the Intel® MPSS distribution

The Intel® MPSS distribution can be obtained from the Intel® Developer Zone.

Untar the Intel® MPSS package:

```
[host]$ tar xvf mpss-3.8-linux.tar
[host]$ cd mpss-3.8
```

As described in Section 1.3.3, we refer to the directory into which the untarred files are placed as $MPSS38.

3.4.2 Uninstalling previous Intel® MPSS installation

Yum and zypper both support software upgrades and downgrades. However, it is necessary that Intel® MPSS upgrades and downgrades be carried out by first completely uninstalling existing software stack components, followed by a clean installation of the replacement software.

1. To check for a previously installed version of Intel® MPSS package:

```
[host]$ rpm -qa | grep -e intel-mic -e mpss
```

Skip to Section 3.4.4 if there is no previous installation.

Unload the software stack driver:

```
[host]# service mpss unload
```

Uninstall the software stack.

- To uninstall 3.x-based builds:

```
[host]$ cd $MPSS3X
[host]$ ./uninstall.sh
```

- To uninstall the pre-3.x builds:
3.4.3 Rebuilding Intel® MPSS host drivers

Both Red Hat® and SUSE® release minor kernel version updates. Kernel update may create version incompatibilities with the Intel® MPSS® host and InfiniBand® drivers, preventing them from loading.

To determine if your host kernel has been updated, you can execute:

```
[host]$ uname -r
```

Compare the returned value to the default versions listed in Table 2. If your host kernel is not updated proceed to Section 3.4.4. Otherwise it may be required to rebuild Intel® MPSS drivers for proper execution.

**Red Hat® Enterprise Edition (RHEL*)**

1. Ensure the prerequisites are installed:

```
[host]# yum install kernel-devel
```

2. Regenerate the Intel® MPSS driver module package:

```
[host]$ cd $MPSS38/src/
[host]$ rpmbuild --rebuild mpss-modules*.src.rpm
```

3. The resulting mpss-modules binary RPMs are located by default at $HOME/rpmbuild/RPMS/x86_64. Copy the mpss-modules RPMs to the modules directory:

```
[host]$ cd $HOME/rpmbuild/RPMS/x86_64
[host]$ cp mpss-modules*`uname -r`*.rpm $MPSS38/modules
```

4. Proceed to Section 3.4.4.

**Suse® Linux® Enterprise Server (SLES*)**

1. Ensure the prerequisites are installed:

```
[host]# zypper install kernel-default-devel rpm-build
```

2. Regenerate the Intel® MPSS driver module package:

```
[host]$ cd $MPSS38/src/
[host]$ rpmbuild --rebuild mpss-modules*.src.rpm
```
3. The resulting *mpss-modules* binary RPMs are located by default at /usr/src/packages/RPMS/x86_64 or $HOME/rpmbuild/RPMS/x86_64 (depending on the OS version). Copy the *mpss-modules* RPMs to the *modules* directory:

```
[host]$ cd /usr/src/packages/RPMS/x86_64
[host]$ cp mpss-modules`uname -r`.rpm $MPSS38/modules
```

4. Proceed to Section 3.4.4.

### 3.4.4 Installing Intel® MPSS

1. The modules directory contains packages for all supported host OS kernels, including packages that were rebuilt in Section 3.4.3. Copy the modules corresponding to your host kernel to $MPSS38:

```
[host]$ cd $MPSS38
[host]$ cp ./modules/*`uname -r`.rpm .
```

2. Install the software stack:

   - Red Hat® Enterprise Linux®
     ```
     [host]# yum install *.rpm
     ```
   - SUSE® Linux® Enterprise Server
     ```
     [host]# zypper install *.rpm
     ```

   **Note:** Intel® MPSS packages are not GPG signed. If local package GPG checking is enabled in the *yum.conf*, or *zypp.conf* file, use the command below to install the software stack:

   ```
   • Red Hat® Enterprise Linux®
     [host]# yum install --nogpgcheck *.rpm
   • SUSE® Linux® Enterprise Server
     [host]# zypper --no-gpg-checks install *.rpm
   ```

3. Load the *mic.ko* driver:

   ```
   [host]# modprobe mic
   ```

### 3.4.5 Updating the coprocessor’s flash and SMC firmware

After the base software stack is installed, it is strongly recommended to update each coprocessor’s flash and SMC firmware to the version distributed with the software stack installation. The $MPSS38/docs/readme.txt file lists the versions of the Flash and SMC firmware in the distribution.

Running Intel® MPSS with incorrect Flash or SMC firmware versions is not supported and may lead to erratic behavior.
These steps will not work if the flash files (ending in `.rom.smc`) are moved to a location other than the default installation path.

To verify what version of the flash is installed execute:

```
[host]# micflash -getversion
```

The current flash version must be >= 375.

Check the status of each coprocessor:

```
[host]$ micctrl -s
```

The `micctrl` utility is installed in `/usr/sbin`, which in SLES* is not by default included in the user's `PATH` variable. Modify your `PATH` variable or provide an absolute path to `micctrl` (`/usr/sbin/micctrl`) in order to execute above command.

If the status for all of the coprocessors shows 'ready', go to step 2; otherwise, set the coprocessor(s) to a 'ready' state:

```
[host]# micctrl -rw
```

1. Determine the stepping and board SKU of each coprocessor to be updated. The `micinfo` utility can be used if this information is not already known. For example:

```
[host]# micinfo -group Board
```

    Board
    ---
    | Vendor ID   | 0x8086 |
    | Device ID   | 0x225d |
    | Subsystem ID| 0x2500 |
    | Coprocessor Stepping ID | 1 |
    | PCIe Width   | x16 |
    | PCIe Speed   | 5 GT/s |
    | PCIe Max payload size | 256 bytes |
    | PCIe Max read req size | 512 bytes |
    | Coprocessor Model | 0x01 |
    | Coprocessor Model Ext | 0x00 |
    | Coprocessor Type | 0x00 |
    | Coprocessor Family | 0x0b |
    | Coprocessor Family Ext | 0x00 |
    | **Coprocessor Stepping** | **B0** |
    | **Board SKU** | **ES2-A1330** |
    | ECC Mode | NotAvailable |
    | SMC HW Revision | NotAvailable |

**Note:** Some data is not available while the coprocessor is not booted.

2. Update the flash image.

    a. If the coprocessor to be updated is any C0 stepping SKU, or a 5110P B1 SKU with a TA of G65758-253 or higher (for 5110P B1 SKUs, the TA is on a sticker affixed to the coprocessor) then execute:

    ```
    [host]# micflash -update -device all
    ```
The command above will update all installed coprocessors. To update a specified coprocessor micN, execute:

```bash
[host]# micflash -update -device N
```

**For example:**

```bash
[host]# micflash -update -device 0
No image path specified - Searching: /usr/share/mpss/flash
mic0: Flash image: /usr/share/mpss/flash/EXT_HP2_B1_0390-02.rom.smc
mic0: Flash update started
mic0: Flash update done
mic0: SMC update started
mic0: SMC update done
mic0: Transitioning to ready state
```

Please restart host for flash changes to take effect

b. Otherwise, execute:

```bash
[host]# micflash -update -device all -smcbootloader
```

This will update all installed coprocessors. To update coprocessor micN, execute:

```bash
[host]# micflash -update -device N -smcbootloader
```

**For Example:**

```bash
[host]# micflash -update -device 0 -smcbootloader
No image path specified - Searching: /usr/share/mpss/flash
mic0: Flash image: /usr/share/mpss/flash/EXT_HP2_B0_0390-02.rom.smc
mic0: SMC boot-loader image: /usr/share/mpss/flash/EXT_HP2_SMC_Bootloader_1_8_4326.css_ab
mic0: SMC boot-loader update started
mic0: SMC boot-loader update done
mic0: Transitioning to ready state
mic0: Flash update started
mic0: Flash update done
mic0: SMC update started
mic0: SMC update done
mic0: Transitioning to ready state
```

Please restart host for flash changes to take effect

3. Reboot the host system for all flash and SMC changes to take effect. Be sure to wait for the flash update to complete before rebooting.

4. You will validate the flash update in [Section 3.4.8](#).

`mpssflash` is a POSIX-compliant version of `micflash`. For detailed information about `micflash` or `mpssflash`, refer to their man pages:
3.4.6 Initializing Intel® MPSS default configuration settings

Intel® MPSS configuration is based on parameters contained in several configuration files. The parameters in /etc/mpss/default.conf are treated as common to all coprocessors in the system. In addition, each coprocessor has an associated /etc/mpss/micN.conf configuration file, where N is an integer number (0, 1, 2, ..., 255) that identifies a coprocessor. Each parameter in a coprocessor specific file takes precedence in configuring the corresponding coprocessor, overriding default.conf if the same parameter is also included in that file.

The micctrl --initdefaults command creates these files if they do not already exist, and populates them with default parameter values. In addition, if a previous configuration file exists, but some parameter is not configured, this command will add a default configuration value.

The micctrl --initdefaults command should be performed after the initial Intel® MPSS installation and after each subsequent installation of a new software stack release.

The micctrl --initdefaults command will not change existing configuration settings, with the following exception: The Intel® MPSS configuration file format is versioned, with the version indicated by a Version parameter in the configuration file. If a configuration already exists, then micctrl --initdefaults will update the configuration format if necessary. The semantics of the updated configuration should be invariant.

Some users switch between different versions of the software stack. When this is the case, and because micctrl --initdefaults does not know how to downgrade a configuration from a newer to an older format, it is recommended to make a copy of the configuration files before calling micctrl --initdefaults.

The default configuration produced by the micctrl --initdefaults command is sufficient for users who will be using the offload programming model on a workstation. You can view a summary of the current configuration parameters with:

[host]$ micctrl --config

The following is typical of the default configuration:

```
mic0:
=================================================================
Config Version: 1.1
Linux Kernel: /usr/share/mpss/boot/bzImage-knightscorner
BootOnStart: Enabled
Shutdowntimeout: 300 seconds
ExtraCommandLine: highres=off
PowerManagment: cpufreq_on;corec6_off;pc3_on;pc6_off
Root Device: Dynamic Ram Filesystem /var/mpss/mic0.image.gz
from:
```
Base: CPIO /usr/share/mpss/boot/initramfs-knightscorner.cpio.gz
CommonDir: Directory /var/mpss/common
Micdir: Directory /var/mpss/mic0
Network: Static Pair
  Hostname: snhondo-desktop7-mic0.dd.domain.com
  MIC IP: 172.31.1.1
  Host IP: 172.31.1.254
  Net Bits: 24
  NetMask: 255.255.255.0
  MtuSize: 64512
  MIC MAC: 4c:79:ba:15:00:1e
  Host MAC: 4c:79:ba:15:00:1f
Cgroup:
Memory: Disabled
Console: hvc0
VerboseLogging: Disabled
CrashDump: /var/crash/mic 16GB

The micctrl tool can be used to modify the configuration, and it is also possible to modify the configuration by directly editing the Intel® MPSS configuration files. Section 3.7.8.2 contains an overview of the configuration process, while later sections discuss a variety of configuration tasks.

### 3.4.7 Starting Intel® MPSS

With Intel® MPSS installed and the flash and SMC versions up-to-date, the coprocessor(s) can be booted. To boot the coprocessor(s), execute the following:

```bash
[host]# service mpss start
```

Starting the mpss service launches the mpssd daemon, and also boots the installed coprocessor(s) if the BootOnStart config option is set to enabled (default).

The following command configures the mpss service to start when the host OS boots:

```bash
[host]# chkconfig mpss on
```

This command disables the mpss service from starting when the host OS boots:

```bash
[host]# chkconfig mpss off
```

See the chkconfig man page for details.

**Note:** On RHEL* 7 and SLES* 12, starting the mpss service (systemctl start mpss) on a system with a large number of coprocessors can take longer than the default value of five minutes of the TimeoutSec parameter in the /etc/systemd/system/mpss.service file. In this case it is necessary to increase TimeoutSec to a larger value.
3.4.8 Validating Intel® MPSS installation

Having booted all the coprocessors in the system, Intel® MPSS provides utilities that can be used to perform basic tests to validate that the installation was performed correctly.

Appendix I provides troubleshooting advice in the event that problems are encountered during installation.

3.4.8.1 Logging in to the coprocessor using SSH

Users may validate the network setup before attempting to connect to a coprocessor. Running `ifconfig` command should indicate that network interfaces for coprocessors are up. Presented below is example output of the `ifconfig` command run on a system with a single coprocessor:

```
[host]# ifconfig
eth0 <Output truncated>
mic0 Link encap:Ethernet  HWaddr 4C:79:BA:12:12:8B
    inet addr:172.31.1.254  Bcast:172.31.1.255
    Mask:255.255.255.0
    inet6 addr: fe80::4e79:baff:fe12:128b/64 Scope:Link
    UP BROADCAST RUNNING MTU:64512  Metric:1
    RX packets:94 errors:0 dropped:0 overruns:0 frame:0
    TX packets:159 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
    RX bytes:11945 (11.6 Kb)  TX bytes:17781 (17.3 Kb)
```

If `micN` is not shown among network interfaces revisit Network Manager configuration in Section 3.1.8.

At this point you should be able to SSH into a coprocessor. For example, to SSH into `mic0`:

```
[host]$ ssh mic0
[mic0]$
```

If the following message appears:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

remove the mic0 RSA from the user’s `known_hosts` file, typically found in the `$HOME/.ssh/` directory, and then try again.

**Note:** When running an `ssh` command in the background, it is preferable to use the `-f` option instead of appending "&" to the command:

```
[host]$ ssh -f hostname "sleep 20; echo complete"
```
3.4.8.2 Validating installation using Intel® MPSS tools

`miccheck` performs sanity checks by running a suite of diagnostic tests. The following example shows `miccheck` output after a successful software stack installation:

```
[host]$ miccheck
MicCheck 3.8-r1
Copyright 2013 Intel Corporation All Rights Reserved
Executing default tests for host
  Test 0: Check number of devices the OS sees in the system ... pass
  Test 1: Check mic driver is loaded ... pass
  Test 2: Check number of devices driver sees in the system ... pass
  Test 3: Check mpssd daemon is running ... pass
Executing default tests for device: 0
  Test 4 (mic0): Check device is in online state and its postcode is FF ... pass
  Test 5 (mic0): Check ras daemon is available in device ... pass
  Test 6 (mic0): Check running flash version is correct ... pass
  Test 7 (mic0): Check running SMC firmware version is correct ... pass
Status: OK
```

Additionally, the `micinfo` tool provides information about the host and coprocessor hardware and software. Certain device information is only available when executing `micinfo` with root privileges:

```
[host]# micinfo
```

See the `miccheck` and `micinfo` man pages for additional information.

3.4.9 Running “Hello world”

After the coprocessors have been booted, execute a simple program several different ways.

3.4.9.1 “Hello world” native execution using gcc

The point of this exercise is to demonstrate the simplicity with which code can be compiled and run on the coprocessor. The example presented below is a simple code written in standard C. The gcc cross compiler that is included in the Intel® MPSS cross-compilation SDK is used. The compiler is installed at `/opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin/k1om-mpss-linux/k1om-mpss-linux-gcc`. Cross compiling using the SDK is described in detail in Section 8.1.

```
[host]$ cat hello_world.c
#include <stdio.h>
#include <stdlib.h>
void
main()
```
Next, copy the code to the file system on the coprocessor using `scp`:

```
[host]$ scp hello_world mic0:
hello_world      100%   10KB  10.2KB/s   00:00
```

Invoke the application on the coprocessor:

```
[host]$ ssh mic0 /home/<USER>/hello_world
Hello World
```

### 3.4.9.2 "Hello world" native execution using the Intel® C Compiler

This example is very similar to the previous one, but this time using Intel® C Compiler. Note that you will need to install the Intel® Compiler suite to build this example (and the follow-on example highlighting offload directives.) See [Intel® C and C++ Compilers](#) for details on licensing and installation.

Notice that the Intel® C compiler (`icc`) is used with an additional flag (`-mmic`) to indicate that the target architecture is the coprocessor.

```
[host]$ cat hello_world.c
#include <stdio.h>
#include <stdlib.h>
void
main()
{
    printf("Hello World \n");
}
[host]$ icc -mmic hello_world.c -o hello_world
```

Next, copy the code to the file system on the coprocessor using `scp`

```
[host]$ scp hello_world mic0:
hello_world      100%   10KB  10.2KB/s   00:00
```

Invoke the application on the coprocessor.

```
[host]$ ssh mic0 /home/<USER>/hello_world
Hello World
```

#### 3.4.9.2.1 "Hello world" via compiler based offload directives

This example demonstrates the use of offload directives to run code on the coprocessor.

```
[host]$ cat hello_offload.c
#include <stdio.h>
#include <stdlib.h>
void
main()
```
{
    #pragma offload target (mic:0)
    {
        printf("hello_world from offloaded code running on the coprocessor \n");
    }
}

To build it, use the Intel® C compiler, as before with -offload flag.

[host]$ icc -offload hello_offload.c -o hello_offload

Finally, to run it, simply invoke the host-side binary.

[host]$ ./hello_offload
   hello_world from offloaded code running on the coprocessor

### 3.5 Summary of the installation process

At this point the host and coprocessors are configured in the static pair networking configuration. In this configuration, a separate private network was created between the host and each coprocessor. As demonstrated in the previous "Hello World" examples, this configuration supports both the Offload and Native programming models as described in Section 2.2.1.

For users who will be developing and/or executing only Intel® C++/FORTRAN offload directive based programming, basic installation is now complete. You may, however, want to consult Section 6 to learn more about user credentials.

Users who will be performing Native program execution on a standalone platform might also wish to learn more about NFS mounting some or all of the coprocessor file system: see Section 3.6.4 for a discussion on the tradeoffs of local vs. remote file system mounts. Building and adding software to the coprocessor's file system may also be of interest: see Sections 7 and 8.

### 3.6 Basic network configuration

This section touches briefly on criteria for choosing a network configuration type. It is primarily of interest to cluster administrators and those with advanced workstation programming models.

Intel® MPSS supports an Internal Bridge configuration, in which the host and all coprocessors host are on a single network. The External Bridge configuration is more relevant for a cluster environment. This configuration was briefly introduced in Section 2.2.3.2.2.

There are several important considerations.

#### 3.6.1 MAC address assignment

Because the coprocessor networking is based on a Virtual Ethernet driver, MAC addresses of network endpoints must be generated locally. There are several options
available including automatic generation by Intel® MPSS drivers based on device serial number, and direct assignment of an externally specified address. Automatic MAC address generation is sufficient for most configurations. More information on MAC address assignment can be found in Section 5.1.

3.6.2 IP address considerations for external bridging

In an external bridge configuration, IP addresses of all endpoints on the network, including the bridge itself, and all coprocessor endpoints, must be on the same subnet. Generally speaking, IP addresses can be assigned statically by editing appropriate configuration files, or appropriate configuration of a DHCP server made available on the local network. In either case, in a cluster environment it is usually desirable for the IP address of each endpoint to remain static over time so that there is easy correlation of IP address to node.

Local cluster site administrators will want to adopt an IP address assignment pattern that is amenable for their datacenter and local network configurations. To illustrate one example scheme, the following highlights a scenario with two coprocessors installed per host. In this case, a simple IP ordering scheme is used to organize the host bridge interfaces and endpoints within the same subnet such that the IP address of the coprocessors can be the IP address of the host/bridge +1 and +2 respectively.

```
172.31.0.1     node0-eth0
172.31.0.2     node0-mic0
172.31.0.3     node0-mic1
172.31.0.4     node1-eth0
172.31.0.5     node1-mic0
172.31.0.6     node1-mic1
172.31.0.7     node2-eth0
```

A DHCP server can be configured to assign persistent static IP addresses to clients. This can be done by directly editing DHCP server configuration files. Some cluster manager utilities (for instance Warewulf) can also perform such DHCP assignments. Configuring the DHCP server either indirectly through the cluster manager or by directly editing its configuration files is beyond the scope of this document.

3.6.3 Configuring a basic external bridge

This section describes one approach to configuring the coprocessors and host as an external bridge, enabling coprocessors to communicate with other nodes in a cluster. The goal of this section is to configure the network such that InfiniBand® installation can be validated. There are various options for configuring an external bridge which are described in more detail in Section 5.

**Note:** You must manually add a gateway to the `br0` configuration file.

Before you can change the network configuration, you must stop the `mpss` service:

```
[host]# service mpss stop
```

Assuming the IP address distribution shown above, an external bridge, `br0`, on node0 can be configured as:
intel® xeon phi™ coprocessor installation process

[host]# micctrl --addbridge=br0 --type=external --ip=172.31.0.1
[host]# micctrl --network=static --bridge=br0 --ip=172.31.0.2

and on node1 as:

[host]# micctrl --addbridge=br0 --type=external --ip=172.31.0.4
[host]# micctrl --network=static --bridge=br0 --ip=172.31.0.5

micctrl does not slave the physical Ethernet endpoint, for example eth0, to the bridge. This must be done by the administrator by editing the Ethernet configuration file(s). For example, on RHEL*, the eth0 Ethernet configuration file, /etc/sysconfig/network/ifcfg-eth0, should typically have the following content:

DEVICE=eth0
NM_CONTROLLED=no
ONBOOT=yes
BRIDGE=br0
MTU=1500

On SLES* host platforms, the physical port name must be added to the BRIDGE_PORTS entry in the /etc/sysconfig/networks/ifcfg-br0 configuration file, for example:

BRIDGE_PORTS='eth0 mic0 mic1'

At this point the network service must be restarted:

[host]# service network restart

Now start the coprocessors:

[host]# service mpss start

Communication with coprocessors on other nodes on the network should now be possible.

3.6.4 Defining and implementing exported/mounted file systems

As mentioned earlier, the coprocessor’s root file system can be supported in its memory, remotely mounted via NFS, or a combination of the two. For example, sections of the file system that are common across multiple coprocessors might be mounted from a common export on a remote node such as a cluster’s head node; in this case an external bridge is required. Some files might be coprocessor specific; these files can be exported from the local host. It may also make sense to locate certain files in coprocessor memory in order to minimize access latency.

The coprocessor’s operating system includes a virtio block device (virtblk), which uses the Linux® virtio data transfer mechanism to implement a block device. Virtblk can store data on the host in a regular file, Logical Volume Manager volume, or a designated physical device. Virtblk is expected to exhibit lower latency than NFS mounted exports from the host.

One advantage of both NFS and virtblk file systems is persistence. That is, changes to virtblk or NFS mounted files can persist after the coprocessors are shutdown, whereas changes to files in coprocessor memory are lost unless steps are taken to capture them.
Another advantage of NFS and virtblk file systems is capacity. Not only is the ram file system limited by available memory, but allocating coprocessor memory makes it unavailable to application processes executing on the coprocessor.

Only NFS supports sharing of files among multiple devices. Sharing the same file to hold the virtblk file system of more than one coprocessor is not supported by Intel® MPSS.

The following table summarizes the characteristics of the available file systems classes.

<table>
<thead>
<tr>
<th>Latency</th>
<th>Persistence</th>
<th>Sharing</th>
<th>Capacity</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM FS</td>
<td>Smallest</td>
<td>No</td>
<td>No</td>
<td>Small</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reduces memory available to app</td>
</tr>
<tr>
<td>VIRTBLK</td>
<td>Medium</td>
<td>Yes</td>
<td>No (not supported)</td>
<td>Large</td>
</tr>
<tr>
<td>NFS</td>
<td>Largest</td>
<td>Yes</td>
<td>Yes (but not cache coherent)</td>
<td>Large</td>
</tr>
</tbody>
</table>

NFS mounting is discussed throughout Section 4. Configuring the virtio block device is discussed in Section 9.6.

3.6.5 Configuring the host firewall

Client services running on the coprocessor need access to services on a host. If a host firewall is enabled, it may need to be configured to allow access to these services.

3.6.5.1 NFS client access

NFS can be used to mount host exports on the coprocessor. NFS generally requires five services to be running:

- portmapper
- nfsd
- mountd
- lockd
- statd

Of these, at least portmapper, nfsd and mountd must be accessible to the coprocessor’s NFS client through the firewall to enable basic NFS operation. Access to the lockd and statd ports is needed if file locking is required.

The ports for the portmapper and nfsd are statically assigned as follows:

- tcp/udp port 111 - RPC 4.0 portmapper
- tcp/udp port 2049 - nfs server
The ports for the other services are normally dynamically assigned. For firewall considerations, it may be desirable to statically assign ports to these services.

Consult documentation for your host operating system for instructions on static port assignment, and for instructions on allowing access to the NFS services ports through the firewall.

### 3.6.5.2 Other port access considerations

As described in Section 7.2.1.3, zypper can be used on the coprocessor to install RPMs in a repository on the host. That section suggests using the python SimpleHTTPServer. The port which the server uses (8000 by default) must be accessible through the firewall.

Similarly, the ports used by the Ganglia daemon on the host (see Appendix F.1.2) may need to be exposed through the firewall.

Consult documentation for your host operating system for instructions on allowing access to these ports.

### 3.6.6 Installing Lustre* on the coprocessor

The following two RPMs should be installed on the coprocessor:

- `lustre-client-modules-<version>.klom.rpm`
- `lustre-client-<version>.klom.rpm`

Refer to Section 7 for instructions on adding software to the coprocessor’s file system. After the installation is completed the Lustre* is ready for configuration and use.

### 3.6.6.1 Configuring Lustre*

Execute the following commands on the coprocessor to configure the Lustre* client over Virtual Ethernet:

```
[micN]# echo 'options lnet networks="tcp0(micN)"' > /etc/modprobe.d/lustre.conf
modprobe lustre
```

This step assumes that IPoIB is correctly configured on the coprocessor. Please refer to Section 5.4 for instructions.

In this example, the coprocessor’s IPoIB interface is `/ib0`.

```
[micN]# echo 'options lnet networks="o2ib0(ib0),tcp0(micN)"' > /etc/modprobe.d/lustre.conf
```

If you would like to make this configuration persistent across all coprocessor reboots execute the following on the host:

```
[host]# mkdir -p /var/mpss/micN/etc/modprobe.d
[host]# echo 'options lnet networks="o2ib0(ib0),tcp0(micN)"' > /var/mpss/micN/etc/modprobe.d/lustre.conf
```

After `mpss` service restart this configuration will be deployed to the coprocessor.
3.6.6.2 Using Lustre

After completing configuration you can mount Lustre* FS share from your network. You can execute the following commands on the coprocessor:

```
[micN]# mkdir -p /mnt/lustre \
/sbin/mount.lustre <MGS IP>@tcp0:/<lustreFS name> /mnt/lustre
```
or

```
[micN]# /sbin/mount.lustre <MGS IP>@o2ib0:/<lustreFS name> \
/mnt/lustre
```

If you like to make this mount point persistent across all coprocessor reboots execute the following on the host:

```
[host]# mkdir -p /var/mpss/mic0/mnt/lustre
```

afterwards you can add this mount point to `/etc/fstab` for automatic mount.

3.7 Installing OFED

The coprocessors can communicate with external compute nodes over high-bandwidth InfiniBand* fabric provided a supported Intel® True Scale or Mellanox* InfiniBand* host adapter is installed in the platform. This section describes how to install the OFED components that support these capabilities.

Instructions in this sections assume that Intel® MPSS is installed on your system.

**Option 1:**

The Offload computing model is characterized by MPI communication only between the host processors in a cluster. In this model the coprocessors are accessed exclusively through the offload capabilities of products like the Intel® C, C++, and Fortran Compilers, and the Intel® Math Kernel Library (MKL). This mode of operation does not require CCL, and therefore the OFED version in a Red Hat* or SUSE* distribution can be used.

**Option 2:**

If MPI ranks are to be executed on the coprocessors, and if it is required that these ranks communicate directly with an InfiniBand* adapter, then the following installation should be performed. The `ibscif` virtual adapter will provide the best host-to-coprocessor and coprocessor-to-coprocessor transfer performance on systems without an InfiniBand* adapter.
### 3.7.1 Supported OFED distributions

#### Table 5: OFED Distribution vs. Supported Features

<table>
<thead>
<tr>
<th>OFED Distribution (installation section)</th>
<th>Mlx4 (IPoIB)</th>
<th>Mlx5 (IPoIB)</th>
<th>PSM-Direct</th>
<th>ccl-proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSR (cf. 3.7.2)</td>
<td>No (no)</td>
<td>No (no)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OFED-3.12-1 (cf. 3.7.3)</td>
<td>Yes (yes)</td>
<td>Yes (no)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OFED 3.18-2 (cf. 3.7.4)</td>
<td>Yes (yes)</td>
<td>Yes (no)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mellanox* OFED 2.4 (cf. 3.7.5)</td>
<td>Yes (yes)</td>
<td>Yes (yes)</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Several different OFED distributions are supported. Select one which matches your hardware and software requirements, and install it using instructions from the accompanying section.

Each OFED distribution supports all, or a subset of the Intel® MPSS compatible OS distros. Refer to the respective OFED’s release notes for the exact list of supported distros.

**Note:** OFED-3.12-1 is deprecated and will not be supported in future releases of the software stack.

### 3.7.2 Installing TSR

TSR (True Scale Release) is the Intel® True Scale Fabric Host Channel Adapter Drivers and Software stack.

**Note:** Only TSR 7.4.1 is supported by Intel® MPSS 3.8.

**Note:** Installing TSR support will replace the OFED components in your standard distribution. This section describes the steps to install Intel® True Scale Fabric Host Channel Adapter Drivers and Software stack (TSR), an enhanced implementation of OFED that supports Intel® True Scale Fabric Host Channel Adapters (HCA), and enables communication between the coprocessor and an Intel® True Scale Fabric HCA. This installation may overlay some of the RDMA/InfiniBand* components in your Red Hat* or SUSE* distribution. As a result, the Linux* kernel will not load kernel mode software that was built against the Red Hat* or SUSE* RDMA/InfiniBand* software in your distribution. This may require that you rebuild such software against the installed OFED, or obtain a version of the software that was so built. For example, an implementation of the Lustre* file system that was built against a Red Hat* or SUSE* distribution will not be loaded by the Linux* kernel, and must be rebuilt against the installed OFED.
**Note:** This section describes the steps to install Intel® True Scale Fabric Host Channel Adapter Drivers and Software stack (TSR), an enhanced implementation of OFED that supports PSM-Direct. PSM-Direct by default enables direct communication between the coprocessor and an Intel® True Scale Fabric HCA—no additional installation steps are required.

User mode applications will not need to be rebuilt due to this installation.

The following installation should be performed on each compute node containing an Intel® True Scale Fabric HCA.

**Note:** When running MPI in Symmetric mode with more than 16 processes per node, `PSM_RANKS_PER_CONTEXT=<value>` needs to be specified (the value can be 2, 3 or 4; the default value is 1) so that the available 16 contexts can be shared by the ranks.

Intel® True Scale Fabric Host Channel Adapter Drivers and Software (TSR), including the PSM library, is available as a free download from [http://downloadcenter.intel.com](http://downloadcenter.intel.com). It contains OFED software bug fixes and enhanced performance.

1. Go to [http://downloadcenter.intel.com](http://downloadcenter.intel.com)

2. Under Search Downloads, type True Scale and hit Enter.

3. Narrow down the results by selecting the appropriate operating system.

4. Select the version of Intel® True Scale Fabric Host Channel Adapter Host Drivers and Software that supports your operating system. Details of the operating system support can be found in the Release Notes (pdf).

5. Download the appropriate “Intel True Scale Fabric Host Channel Adapter Host Drivers & Software” file as well as the related publications file, which contains documentation.

6. Untar the IntelIB-*.tgz file and access the created directory.

   ```
   [host]# tar xvf IntelIB-*.tgz
   [host]# cd IntelIB-`
   ```

7. Read the README.txt file and install the OFED stack. Detailed installation instructions are available in the Installation Guide (`IFS_FabricSW_InstallationGuide*.pdf`), which is contained in the `Publications_HCA_SW*.zip` archive.

   ```
   [host]# less README.txt
   [host]# ./INSTALL -a --with-xeon-phi
   ```

8. If `yum` was used to install the Intel® MPSS, it is necessary to remove `infinipath-libs` and `infinipath-devel`:

   ```
   [host]# rpm -e --nodeps --allmatches infinipath-libs \infinipath-devel
   ```

9. Install required PSM (Performance Scaled Messaging) libraries and drivers:
[host]$ cd $MPSS38

- Red Hat® Enterprise Linux®

[host]# yum install psm/*.rpm

- SUSE® Linux® Enterprise Server

[host]# zypper install psm/*.rpm

10. Reboot the host as recommended in the Installation Guide. This operation completes the TSR installation procedure. Refer to Section 3.7.6 for instructions on running the software.

### 3.7.3 Installing OFED-3.12-1

**Note:** OFED-3.12-1 is deprecated and will not be supported in future releases of the software stack.

1. Download the distribution tarball from:

   http://downloads.openfabrics.org/OFED/ofed-3.12-1/

2. Untar `OFED-3.12*.tgz` and access the created directory.

   [host]# tar xvf OFED-3.12*.tgz
   [host]# cd OFED-3.12*

3. Install the OFED stack as instructed in OFED README.txt.

   [host]# less README.txt
   [host]# perl install.pl --with-xeon-phi

### 3.7.4 Installing OFED 3.18-2

1. The official distribution tarball can be downloaded by following the link below.


2. Untar `OFED-3.18-2.tgz` and access the created directory.

   [host]# tar xvf OFED-3.18-2.tgz
   [host]# cd OFED-3.18-2

3. Install the OFED stack as instructed in OFED README.txt.

   [host]# less README.txt
   [host]# perl install.pl --with-xeon-phi

**Note:** If your host OS kernel version is older than v3.10 it is required to modify the `/etc/modprobe.d/ibscif.conf` file so that it contains a line “options ibscif new_ib_type=1”.

3.7.5 Installing Mellanox* OFED 2.4

1. Download Mellanox* OFED from:

2. Untar the downloaded archive, read the documentation, and follow the normal installation procedure.

3. Install Intel® MPSS OFED ibpd RPM:
   [host]# rpm -U ofed/ofed-ibpd*.rpm

4. From the $MPSS38/src/ folder of the Intel® MPSS installation, compile dapl, libibscif, and ofed-driver source RPMs:
   [host]# cd $MPSS38/src
   [host]# rpmbuild --rebuild dapl*.src.rpm libibscif*.src.rpm ofed-driver*.src.rpm

5. Install the resultant RPMs:
   • Red Hat* Enterprise Linux*
     [host]# rpm -U ~/rpmbuild/RPMS/x86_64/dapl*rpm
     [host]# rpm -U ~/rpmbuild/RPMS/x86_64/libibscif*rpm
     [host]# rpm -U ~/rpmbuild/RPMS/x86_64/ofed-driver*rpm
   • SUSE* Linux* Enterprise Server:
     [host]# rpm -U /usr/src/packages/RPMS/x86_64/dapl*rpm
     [host]# rpm -U /usr/src/packages/RPMS/x86_64/libibscif*rpm
     [host]# rpm -U /usr/src/packages/RPMS/x86_64/ofed-driver*rpm

3.7.6 Starting OFED

1. Ensure that the mpss service is started by using the service command:
   [host]# service mpss status
   If the service is not running, refer to Section 3.4.7 for instructions on starting it.

2. If using Intel® True Scale Fabric HCAs, or using Mellanox* InfiniBand* adapters and/or the ibscif virtual adapter, start the IB and HCA services by issuing the following:
   [host]# service openibd start

3. If needed, start an opensm service to configure the fabric:
   [host]# service opensm start
   If using Intel® True Scale Fabric HCAs and Intel® True Scale Fabric Switches, it is recommended to use the Intel® Fabric Manager, rather than the opensm. Visit http://www.intel.com/infiniband for information on the fabric management and software tools, downloads, and support contacts.
4. If using CCL-Direct and IPoIB with Mellanox* InfiniBand* adapters, you can enable the IPoIB module to be loaded as part of the ofed-mic service (see Section 5.4) and configure the IP Address and Netmask by editing the /etc/mpss/ipoib.conf file which contains instructions on how to make these changes. See the example ipoib.conf script in Section 5.4.

5. If using Intel® True Scale Fabric HCAs, or using Mellanox* InfiniBand* adapters and/or the ibscif virtual adapter, then start the coprocessor-specific OFED service on the host using:

   [host]# systemctl start ofed-mic

6. The use of ccl-proxy service is applicable only if using Mellanox* InfiniBand* adapters. To start the ccl-proxy service (see configuration in: /etc/mpxyd.conf):

   [host]# systemctl start mpxyd

The use of PSM-Direct, which is applicable only if using Intel® True Scale Fabric HCAs, is enabled by default and does not require starting any services.

### 3.7.7 Stopping/restarting OFED

To stop all OFED support on all variants, stop the following services in the following order:

   [host]# systemctl stop mpxyd
   [host]# systemctl stop opensmd
   [host]# systemctl stop ofed-mic
   [host]# systemctl stop openibd

To restart all OFED components: stop and start them as described above.

### 3.7.8 Validating OFED installation

Several commands are available to validate OFED installation on the host and on the coprocessors.

#### 3.7.8.1 Validating OFED installation on the host

1. `service openibd status` reports whether the driver is loaded.

   [host]# service openibd status

2. `ibv_devices` and `ibv_devinfo` will report which devices are present and whether ports are up or down.

   [host]# ibv_devices
   [host]# ibv_devinfo

3. `ofed_info` reports the OFED version that is installed including installed packages.

   [host]# ofed_info
3.7.8.2 Validating OFED installation on the coprocessor

The same `ibv_devinfo` command can be used to validate OFED installation on a coprocessor after starting the `ofed-mic` service.

On an Intel® True Scale Fabric hosts, the `ibv_devices` command issued on the host will show both `qib0` and `scif_0`, while an `ibv_devices` command issued on the coprocessor will show only `scif_0`.

3.7.8.3 Running an Intel® MPI application.

It is possible to further validate the installation by running an Intel® MPI application.

To run MPI applications using Intel® MPI over tmi² fabric, the `tmi.conf` file should be copied to the Intel® Xeon Phi™ coprocessor using following procedure:

a) Create a directory `etc` in `/var/mpss/common/` directory.

b) Copy the `tmi.conf` file from `<impi_install_dir>/etc64/` directory to `/var/mpss/common/etc`.

c) Start/restart the mpss service.

Refer to the following documents for information on how to install and utilize the library:

- Intel® MPI Library for Linux* OS Installation Guide
- Intel® MPI Library for Linux* OS Reference Manual, section 2.4, *Intel® Xeon Phi™ coprocessor Support*
- Intel® MPI Library for Linux* OS User’s Guide, section 12, *Using the Intel® MPI Library with the Intel® Many Integrated Core (Intel® MIC) Architecture*

---

² Intel MPI natively supports PSM using Tag Matching Interface(TMI)
4 Configuring and booting the coprocessor OS

Like any Linux* based system, booting Linux* on the coprocessor requires a Linux* kernel and a file system image (this document does not cover coprocessor’s firmware because it is not configurable). Because the coprocessor does not have a permanent storage system, these components cannot be installed directly onto it. Instead, they are installed into the host’s file system as part of Intel® MPSS installation. The coprocessor’s kernel command line is constructed based on a set of configuration files on the host and provided to the kernel at boot time. While any of these components can be changed as needed, the most common usage scenarios involve changing the file system image (initramfs) and/or the kernel command line.

The initial file system and kernel command line can be configured by modifying various Intel® MPSS specific files and certain host configuration files. These files can be edited directly or modified using the micctrl utility. Configuration of other software stack components, such as the host driver, is described in later sections of this document.

Section 3.4.5 briefly discussed some basic configuration tasks. In this section coprocessor’s configuration is presented in more detail: which files typically need modification, different approaches and tools to aid configuration, and what happens as a result of setting configuration parameters.

Configuration tasks range from specifying the location of the coprocessor Linux* kernel in the host’s file system, to managing user accounts on the coprocessor and configuring network characteristics. Configuration also includes installing packages into the file system. That topic is additionally covered in Section 7.

Many configuration task can be completed using micctrl. The micctrl utility is a multi-purpose tool that provides two classes of functionality:

- Coprocessor state control – boot, shutdown and reset of attached coprocessors
- Configuration – Some micctrl configuration commands modify parameters in Intel® MPSS-specific configuration files. Other micctrl commands process those configuration files to generate standard Linux* configuration files that replace corresponding ones in the default file system. Still, other micctrl commands modify standard configuration files on the host. The files can also be edited directly.

Using micctrl is referred to as assisted configuration and control, or just assisted configuration, and discussed in Section 4.1.

Alternatively, the coprocessor can also be controlled through its sysfs nodes. Configuration can also be performed by directly editing or otherwise modifying the initial file system image while it is stored on the host or on a coprocessor, and by directly composing coprocessor’s Linux* kernel command line. Similarly, host configuration files can be edited to complete networking and similar configuration requirements. These operations are referred to as manual configuration and control, or just manual control, and discussed in Section 4.2.
### 4.1 Assisted configuration and control

In its most basic form, the assisted configuration process has the following steps:

1. Call `micctrl --initdefaults` after each Intel® MPSS installation to create and/or upgrade a set of configuration files specific for the software stack.

2. Call additional `micctrl` commands to tailor the configuration as necessary.

3. Boot the coprocessors.

For simple configuration tasks, a basic understanding of the usage of `micctrl` configuration commands may be sufficient; the `micctrl` commands are described in detail in Appendix B. For more complex configurations, a deeper understanding of the overall assisted configuration process can be very helpful.

#### 4.1.1 Configuration files

There are several different groups of files that contribute to the final configuration. The following subsections describe these groups, show how and when they are created, and indicate how they are identified.

##### 4.1.1.1 Intel® MPSS-specific configuration files

`micctrl --initdefaults` creates several Intel® MPSS specific configuration files, if they do not already exist, and populates them with default parameter values. There are two primary configuration files of interest here:

- The parameters in `default.conf` are treated as common to all coprocessors in the system.
- There is a `micN.conf` file for each coprocessor in the system. Each parameter in this file takes precedence in configuring the corresponding coprocessor, overriding `default.conf` if the same parameter is in that file. You can think of these as “meta-configuration” files in that they guide the completion of the configuration process.

By default, these files are created in `/etc/mpss`.

Each of these files contains a list of configuration parameters and their arguments. Each parameter must be on a single line. Comments begin with the ‘#’ character and terminate at the first Newline/Carriage return. There are several configuration parameter categories:

1. Parameters that control the coprocessor boot process.
2. Parameters that select coprocessor Linux* kernel to be booted.
3. Parameters that configure the coprocessor file system.
4. Parameters that configure the coprocessor boot command line.
5. Parameters that configure the Virtual Ethernet connection to each coprocessor.
6. Parameters that control some aspects of user accounts.
For example, here is a portion of the contents of *default.conf* when initially created:

```plaintext
# Boot MIC card when MPSS stack is started
BootOnStart Enabled

# Root device for MIC card
RootDevice ramfs /var/mpss/mic0.image.gz

# Control card power state setting
# cpufreq: P state
# corec6: Core C6 state
# pc3: Package C3 state
# pc6: Package C6 state
PowerManagement "cpufreq_on;corec6_off;pc3_on;pc6_off"

Cgroup memory=disabled
```

In this fragment, *BootOnStart* configures the boot process, *RootDevice* defines where the coprocessor file system lives on the host before it is provided to the coprocessor kernel, and *PowerManagement* and *Cgroup* configure the boot command line.

Intel® MPSS-specific configuration file parameters are described in detail in Appendix A.

### 4.1.1.2 Host files

Several networking related host configuration files are optionally created and/or modified by *micctrl* commands. These include `/etc/hosts`, as well as `/etc/sysconfig/network-scripts/ifcfg-micN` on a RHEL* host and `/etc/sysconfig/network/ifcfg-micN` on a SLES* host.

Lines that *micctrl* adds to `/etc/hosts` are appended by the comment "#Generated-by-micctrl".

See Section 5 for details.

### 4.1.1.3 Overlay sets

*micctrl* does not directly modify the installed Intel® MPSS file system image. Instead, one or more file hierarchies overlay corresponding files in the file system image during the boot process. That is, each file in a hierarchy will replace the corresponding file in the base file system image if that file already exists, or will be added to the file system image if the corresponding file does not already exist.

We refer to these hierarchies, collectively, as *overlay sets* or just *overlays*. There are several types of overlay sets.

The configuration files described previously include parameters that point to the various overlay sets described below. Because there can be multiple sets of the software stack configuration files, there can be multiple unique overlay sets.
4.1.1.3.1 Base file system

The overlay process begins with the Base file system. The Base configuration file parameter:

```
Base <type> <location>
```

specifies the file system to be used. <type> can be CPIO to indicate a compressed CPIO archive at <location>, or DIR to indicate an expanded file system hierarchy rooted at <location>. By default, this parameter is in /etc/mpss/micN.conf and set to:

```
Base CPIO /usr/share/mpss/boot/initramfs-knightscorner.cpio.gz
```

See Appendix A.4.2 for details.

The Base parameter can be modified using the `micctrl --base` command:

```
[host]# micctrl --base=<default|cpio|dir> [--new=<location>] \
    [mic card list]
```

or by directly editing Intel® MPSS configuration files. See Appendix B.4.4.1 for details.

4.1.1.3.2 Common overlay set

The common overlay set, by default rooted at /var/mpss/common, can be populated with files that will overlay the file system of all coprocessors. For example, if administrator creates the file /var/mpss/common/etc/foo, it will overlay /etc/foo in the file system of each coprocessor.

The `CommonDir` parameter:

```
CommonDir <commondir>
```

is typically found in the default.conf configuration file, and specifies the common overlay set, where <commondir> is the root of the overlay hierarchy. By default, this parameter is set to:

```
CommonDir /var/mpss/common
```

No files are created in this directory by default. See Appendix A.4.2 for details.

Overlay parameters can be created or modified using the `micctrl --commondir` command:

```
[host]# micctrl --commondir=<commondir> [mic card list]
```

or by directly editing Intel® MPSS configuration files. See Appendix B.4.4.3 for details.

4.1.1.3.3 Per-coprocessor overlay set

`micctrl --initdefaults` creates and populates an overlay set of files for each installed coprocessor. However, if a file already exists, it is not changed. By default, these overlays are rooted at /var/mpss/micN. The per-processor overlay set includes the following files:
/etc
/etc/fstab
/etc/group
/etc/hostname
/etc/hosts
/etc/localtime
/etc/nsswitch.conf
/etc/passwd
/etc/resolv.conf
/etc/shadow
/etc/init.d/
/etc/network/
/etc/network/interfaces
/etc/pam.d/
/etc/pam.d/common-account
/etc/pam.d/common-auth
/etc/pam.d/common-session
/etc/ssh/
/etc/ssh/ssh_host_dsa_key
/etc/ssh/ssh_host_dsa_key.pub
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_rsa_key.pub
/etc/ssh/ssh_host_key
/etc/ssh/ssh_host_key.pub
/etc/ssh/ssh_host_ecdsa_key
/etc/ssh/ssh_host_ecdsa_key.pub
/etc/rc1.d/
/etc/rc5.d
/home
/home/micuser
/home/micuser/.profile
/home/micuser/.ssh
/home/micuser/.ssh/id_dsa
/home/micuser/.ssh/id_dsa.pub
/home/micuser/.ssh/id_rsa
/home/micuser/.ssh/id_rsa.pub
/home/micuser/.ssh/authorized_keys
/root
/root/.profile
/root/.ssh
/root/.ssh/id_dsa
/root/.ssh/id_dsa.pub
/root/.ssh/id_rsa
/root/.ssh/id_rsa.pub
/root/.ssh/authorized_keys
(The .ssh key files are created depending on which key files the user or root has created.)

Thus, for each of the above files, there is a corresponding file rooted at /var/mpss/micN. For example, micctrl --initdefaults creates and initializes the file /var/mpss/mic0/etc/fstab, which, at boot time, will replace /etc/fstab in the file system of coprocessor mic0.

The MicDir parameter in each micN.conf configuration file:

MicDir <micdir>

specifies the coprocessor specific overlay set for the corresponding coprocessor, where <micdir> is the root of the overlay hierarchy. By default, this parameter is set to:

MicDir /var/mpss/micN

for coprocessor micN. See Appendix A.4.2 for details.

Micdir parameters can be created or modified using the micctrl --micdir command:

[host]# micctrl --micdir=<micdir> [mic card list]

or by directly editing Intel® MPSS configuration files. See Appendix B.4.4.3 for details.

4.1.1.3.4 User defined overlay sets

Arbitrary sets of files can be defined by the user to overlay corresponding files in the file systems of one or more coprocessor. The Intel® MPSS configuration file Overlay parameter describes a single such overlay set:

Overlay (Simple|File|RPM) <source> <target> (on|off)

The <Simple,File,RPM> type parameter determines how the contents of the <source> and <target> are interpreted. No Overlay parameters are created by default. See Appendix A.4.2 for details.

Overlay parameters can be created or modified using the micctrl --overlay command or by directly editing Intel® MPSS configuration files:

[host]# micctrl --overlay=<type> --source=<dir> \ [-target=<target>] --state=(on|off|delete) \ [mic card list]

A default.conf or micN.conf configuration file can have multiple overlay parameters. See Appendix A.4.2 for details.

The RPM overlay type is a special case that identifies RPM based packages that are to be installed into one or more coprocessor file systems at boot time. See Section 7.2.1.1 and Appendix B.4.4.5 for details. The mpss-3.8-k1om.tar file is comprised of over 1900 RPM files that were built for installation on the coprocessor.

Note: It is strongly advised not to do the following:

[host]# micctrl --overlay=RPM --source=$MPSS38_K1OM --state=on
Parsing this command will cause the coprocessor to attempt to install all (over 1900) RPMs from the mpss-3.8-k1om.tar file. In general, care should be taken to install only RPMs that are actually needed.

4.1.1.4 Constructing the file system

`micctrl` constructs the coprocessor file system hierarchy for each coprocessor from the overlay sets described above. The process has the following steps:

1. If the Base file system is in the form of a compressed CPIO archive, it is first decompressed and extracted to a temporary location, before files are overlaid.
2. The Base file system is then overlaid by the CommonDir overlay set.
3. The resulting hierarchy is overlaid by any hierarchies indicated by Overlay parameters in `default.conf`.
4. The result is then overlaid by the coprocessor-specific MicDir hierarchy for that coprocessor, as specified by the MicDir parameter in the corresponding `micN.conf` file.
5. The resulting hierarchy is then overlaid by file hierarchies specified by Overlay parameters in the corresponding `micN.conf` file.
6. The resulting hierarchy is re-archived and compressed if the file system will be resident in the coprocessor memory, as specified by the RootDevice RamFS or RootDevice SplitRamFS parameter. It is left as an expanded file hierarchy on the host if it is to be NFS mounted.

4.1.2 Initializing, updating and resetting the configuration files

As discussed previously, `micctrl --initdefaults` is used to create and initialize a set of configuration files. The same `micctrl --initdefaults` command should also be called after installing a revision of Intel® MPSS so that `micctrl` can perform any upgrades to a configuration file set in the event that some software stack configuration file parameters were deprecated. In that case `micctrl` will replace the deprecated parameter with equivalent replacement parameterization. To aid this process, each `micN.conf` configuration file includes a version parameter:

```
Version <major number> <minor number>
```

This parameter should not be manually edited.

`micctrl --initdefaults` can also be parameterized to perform some additional user authentication, network, and coprocessor configuration operations. Refer to Appendix B.4.2.1 for details.

As mentioned earlier, if you wish to use a configuration again, for example with an earlier Intel® MPSS release create a copy of the existing configuration before calling `micctrl --initdefaults`.

Many `micctrl` operations directly modify files in the per-coprocessor overlay file sets (Section 4.1.1.3.2). However network configuration can be a multistep process and directly editing the various network configuration files is not feasible. Instead, network
configuration settings are accumulated in *micN.conf* configuration files, and the accumulated settings are propagated to the per-processor files set and host configuration files.

Several *micctrl* commands are intended to assist the user in the event that a configuration is problematic for some reason. *micctrl --resetdefaults* attempts to restore the configuration parameters and the associated coprocessor file systems back to the default state. It shuts down the current network, removes several files in the /etc directories in the per-coprocessor overlay sets, removes the old configuration files *default.conf* and *micN.conf*, and effectively calls *micctrl --initdefaults*.

*micctrl --resetdefaults* does not remove files that the user has added to the various overlay sets. See Appendix B.4.2.2 for details.

If *micctrl --resetdefaults* fails to resolve configuration problems, *micctrl --cleanconfig* can be called to completely remove all software stack created files and overlay sets, including files that the user has created in an Intel® MPSS per-device or common overlay set. See Appendix B.4.2.3 for details.

### 4.1.3 Micctrl directory path modifiers

*micctrl* supports several directory path modifiers that override the default directory locations that it accesses. These modifiers enable building and maintaining multiple Intel® MPSS configurations.

In the remainder of this document, we almost always assume the default values of these modifiers. That is, we typically describe the *default.conf* and *micN.conf* configuration files as being in the /etc/mpss directory. It should be understood that the correct location of these files is $DESTDIR/$CONFIGDIR/default.conf (see below).

Similarly, we typically assume that per-coprocessor overlay hierarchy is rooted at the default /var/mpss/micN, which is the default value of the *MicDir* configuration parameter.

#### 4.1.3.1 $destdir

We use the symbol $DESTDIR to indicate a directory path that *micctrl* prepends to all accesses of files which it creates. By default $DESTDIR is / . The default can be overridden by assigning a value to *MPSS_DESTDIR* environment variable, for instance:

```
[host]$ export MPSS_DESTDIR=<destdir>
```

The $DESTDIR default and *MPSS_DESTDIR* environment variable can be overridden with the *--destdir=<destdir>* *micctrl* global option.

$DESTDIR is applied dynamically. That is, *micctrl* prepends its current value of to each file path at the time of file access. This means that the same $DESTDIR value must be used consistently to access a particular set of files.

For example, given the following command sequence:

```
[host]# export MPSS_DESTDIR=/destdir1
[host]# micctrl --initdefaults
```
Configuring and booting the coprocessor OS

$micctrl$ will create a new configuration, if one does not exist, rooted at $/destdir1$. However, for the command sequence:

```
[host]# export MPSS_DESTDIR=/destdir1
[host]# micctrl --destdir=/destdir2 --initdefaults
```

$micctrl$ will create a new configuration, if one does not exist, rooted at $/destdir2$ because the $--destdir$ global option overrides the value of $DESTDIR$ that was set by $MPSS_DESTDIR$ environment variable.

When the current value of $DESTDIR$ is not default, $micctrl$ will not make any changes to the host's network configuration. In particular, it will not create network configuration files (For instance: $/etc/sysconfig/network-scripts/ifcfg-micN$), nor will it bring a network interface up or down.

### 4.1.3.2 $configdir$

We use the symbol $CONFIGDIR$ to indicate the directory path at which $micctrl$ creates and/or accesses the default.conf and micN.conf configuration files, and the conf.d configuration directory. By default $CONFIGDIR$ is $/etc/mpss$.

The default can be overridden by defining the $MPSS_CONFIGDIR$ parameter:

```bash
MPSS_CONFIGDIR <confdir>
```

in the $/etc/sysconfig/mpss.conf$ file. For example:

```bash
MPSS_CONFIGDIR /home/mic/configdir
```

**Note:** $/etc/sysconfig/mpss.conf$ is not created by default, and must be created by the user.

The $CONFIGDIR$ default and $MPSS_CONFIGDIR$ parameter can be overridden by assigning a value to the $MPSS_CONFIGDIR$ environment variable, for example:

```
[host]# export MPSS_CONFIGDIR=<confdir>
```

The $CONFIGDIR$ default, $MPSS_CONFIGDIR$ parameter, and $MPSS_CONFIGDIR$ environment variable can be overridden by the $--configdir=<confdir>$ or $-c <confdir>$ $micctrl$ global option.

$CONFIGDIR$ is applied dynamically. That is, $micctrl$ prepends the current $DESTDIR$/CONFIGDIR value to each access of a default.conf or micN.conf configuration file, or conf.d directory. Consequently the same $DESTDIR$/CONFIGDIR value must be used consistently to access a particular set of files.

### 4.1.3.3 $vardir$

We use the symbol $VARDIR$ to indicate the directory path variable at which the $micctrl$ $--initdefaults$ and $--resetconfig$ commands create the common and micN overlay hierarchies, and at which the $micctrl$ $--rootdev$ command places a ramfs file system image or NFS file system hierarchy. By default $VARDIR$ is $/var/mpss$. The default can be overridden by assigning a value to the $MPSS_VARDIR$ environment variable, for example:

```
[host]# export MPSS_VARDIR=<vardir>
```
The `$VARDIR` default and `MPSS_VARDIR` environment variable can be overridden by the `--vardir=<vardir>` suboption to the `micctrl --initdefaults`, `--resetconfig`, and `--rootdev` commands.

`$VARDIR` is applied persistently. That is, when `micctrl --initdefaults` or `--resetconfig` adds or modifies a `CommonDir` or `MicDir` parameter to an Intel® MPSS configuration file, the parameter values has the `$VARDIR` path prepended.

For example, assuming a configuration does not currently exist, then the command sequence:

```
[host]# export MPSS_VARDIR=/vardir1
[host]# micctrl --initdefaults
```

will add the following parameter to `$DESTDIR/$CONFIGDIR/default.conf`:

```
CommonDir /vardir1/common
```

and the following parameters to `$DESTDIR/$CONFIGDIR/mic0.conf`:

```
Micdir /vardir1/mic0
RootDevice Ramfs /vardir1/mic0.image.gz
```

The above paths are not prepended by the value of `$DESTDIR`, which is applied dynamically.

In the above example, `micctrl` will also populate a per-coprocessor overlay set at `$DESTDIR/vardir1/micN` rather than at the default `$DESTDIR/var/mpss/micN`.

### 4.1.3.4 `$srcdir`

We use the symbol `$SRCDIR` to indicate the directory path at which the `micctrl --initdefaults`, `--resetdefaults`, `--resetconfig`, and `--cleanconfig` commands look for the coprocessor’s Linux* kernel image and default file system image. By default `$SRCDIR` is `/usr/share/mpss/boot`. The default can be overridden by assigning a value to the `MPSS_SRCDIR` environment variable, for example:

```
export MPSS_SRCDIR=<srcdir>
```

The `$SRCDIR` default and `MPSS_SRCDIR` environment variable can be overridden by the `--srcdir` suboption to the `micctrl --initdefaults`, `--resetdefaults`, `--resetconfig`, and `--cleanconfig` commands.

`$SRCDIR` is applied persistently. For example, assuming that the `Base` and `OSimage` parameters are not currently defined in the `$DESTDIR/$CONFIGDIR/micN.conf` configuration file, then the command:

```
[host]# micctrl --initdefaults
```

or

```
[host]# micctrl --initdefaults --srcdir=srcdir1
```

adds the following parameters to `$DESTDIR/$CONFIGDIR/micN.conf`:
Base CPIO srcdir1/initramfs-knightscorner.cpio.gz
0Simage srcdir1/bzImage-knightscorner srcdir1/System.map-knightscorner

to the $DESTDIR/$CONFIGDIR/micN.conf configuration file of each specified coprocessor. The above path is not prepended by $DESTDIR, which is applied dynamically.

4.1.3.5 $netdir

We use the symbol $NETDIR to indicate the directory path at which the micctrl --initdefaults, --resetdefaults, --resetconfig, --cleanconfig, --mac, --network, --addbridge, --modbridge and --delbridge commands create and/or edit network control files. By default $NETDIR is /etc/sysconfig/network-scripts on RHEL* host platforms and /etc/sysconfig/network on SLES* host platforms. The default can be overridden by assigning a value to the MPSS_NETDIR environment variable to some value, for example:

[host]$ export MPSS_NETDIR=<netdir>

The $NETDIR default and MPSS_NETDIR environment variable can be overridden by the --netdir suboption to the micctrl --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands.

For example, the command:

[host]# micctrl --network=static --mtu=1500 mic0

or

[host]# micctrl --network=static --netdir=<netdir>\n--mtu=1500 mic0

creates a ifcfg-mic0 network control file in <netdir>, where <netdir> is the current value of $NETDIR.

4.1.4 Boot configuration

To boot the coprocessor, the mpssd daemon needs to:

- Determine the kernel to be booted.
- Identify and/or build the file system image.
- Build the kernel command line.

Parameters in the default.conf and micN.conf files are evaluated for this purpose. The default.conf and micN.conf configuration files to be consulted are determined by the configdir specification hierarchy described earlier in Section 4.1.3.2.

The following sections describe the parameters that are evaluated for this purpose.
4.1.4.1 Specifying the Linux* kernel

The OSimage parameter:

    OSimage <linux_kernel_image> <system_address_map_file>

specifies the coprocessor Linux* OS kernel image and associated system address map file.

By default, this parameter is set to:

    OSimage /usr/share/mpss/boot/bzImage-knightscorner
          /usr/share/mpss/boot/System.map-knightscorner

in the /etc/mpss/micN.conf configuration file of each specified coprocessor.

The micctrl --osimage command:

    micctrl --osimage=<osimage> [mic card list]

can be used to modify the --osimage parameter, or the parameter can be edited directly.

4.1.4.2 Specifying and building the file system image

The RootDevice parameter specifies both where the root file system resides, as well as how and when it is constructed:

    RootDevice <type> <location> [usr_location>

By default, this parameter is set to:

    RootDevice Ramfs /var/mpss/micN.image.gz

in the /etc/mpss/micN.conf configuration file of each specified coprocessor.

When <type> is Ramfs, a compressed cpio ram disk image is first constructed from overlay sets, as described in Section 4.1.1.4, and placed at <location> when a boot request is given. This image is pushed to coprocessor's memory, where it is expanded.

When <type> is StaticRamfs, there must already be a compressed cpio ram disk image at <location>. The specified image will be used without rebuilding when the coprocessor is booted.

If <type> is NFS, the booting coprocessor will mount its root file system from the NFS export specified by <location>. The <location> must be a fully qualified NFS mount location with the format "server:location". At boot time, there must already be a root file system hierarchy at <location>.

If <type> is SplitNFS, the booting coprocessor will mount its root file system, /, from the NFS export specified by <location> and its /usr file system from the NFS export specified by <usr_location>. Both <location> and <usr_location> must be a fully qualified NFS mount locations with the format "server:location". At boot time, there
must already be a root file system hierarchy (minus /usr) at <location>, and a /usr hierarchy at <usr_location>.

The micctrl --rootdev command:

```
    micctrl --rootdev=<type> --target=<location> --server=<name> \
            [--usr=<usr_location> [-c] [-d] [mic card list]]
```

can be used to modify the RootDevice parameter in one or more micN.conf configuration files, or the parameter can be directly edited in a configuration file.

Refer to Appendix B.4.3 for details.

### 4.1.4.3 Building the kernel commandline

The mpssd daemon constructs a kernel command line based on several parameters in the Intel® MPSS configuration files. Most of these are described in Appendix A.3. For each such parameter, there is a corresponding micctrl command that can be used to modify the parameter, or these parameters can be modified directly.

### 4.1.5 Assisted boot process

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

#### 4.1.5.1 Instructing the driver to boot the coprocessor

On many Linux* based systems the grub boot loader loads and executes a Linux* kernel image selected from the grub configuration file. The grub configuration file lists available kernels as well as parameters to be passed through the kernel command line. The mpssd host daemon and Intel® MPSS configuration files play a similar role in directing the coprocessor boot process.

The mpssd daemon first constructs a (partial) kernel command line for each coprocessor being booted, based on parameters in the Intel® MPSS configuration files. These parameters are described throughout this document, and in Appendices A.3 and A.4.1. The resulting command line is written to the /sys/class/mic/micN/cmdline sysfs node, where the mic.ko driver will retrieve it.

Next, mpssd requests that the mic.ko driver start the coprocessor by writing a boot string to the /sys/class/mic/micN/state sysfs node. The format of this string depends on whether the coprocessor file system is to be a RAM file system or is to be NFS mounted.

For the RAM file system, the format is:

```
    boot:linux:<linux_kernel_image>:<ram_disk_file>
```

and for NFS, it is:

```
    boot:linux:<linux_kernel_image>
```

The linux:<linux_kernel_image> part of the boot argument specifies the location of the Linux* image which is used to boot the coprocessor. mpssd obtains this value from the OSimage parameter.
The `<ram_disk_file>` part specifies the file system image. `mpssd` obtains this file name from the `RootDevice` parameter.

When the `mic.ko` host driver receives the boot request, it first verifies whether the coprocessor is in the `ready` state, indicating that it has finished its HW initialization sequence and is ready to receive a kernel and file system image to continue the boot process. If the coprocessor is not ready to boot, the driver will report an error and will not attempt to perform the boot. Otherwise, the coprocessor state is set to `booting`.

Next the `mic.ko` host driver copies the specified Linux* image and file system image to the coprocessor memory and writes the constructed command line via the standard Linux* kernel boot protocol structure.

The driver’s last step is to write to a coprocessor register, effectively instructing it to jump to the provided `bzImage` to finish the kernel boot process.

### 4.1.5.2 The coprocessor OS kernel initial phases

The coprocessor Linux* kernel goes through virtually the same startup process as on any Linux* based machine. It initializes the bootstrap processor, starts kernel services, including various built-in modules, and brings up all the application processors (APs) to full SMP state. The final step in the boot process involves mounting the root file system so that `/init` can be executed.

The initial ram disk image contains the loadable modules required for the real root file system. Some of the arguments passed in the kernel command line are host memory addresses required by those modules. The `init` program parses the kernel command line for needed information and creates an `/etc/modprobe.d/modprobe.conf` file needed by the coprocessor’s `init` process.

In the next step, the `root` command line parameter determines whether `init` mounts the file system image that `mic.ko` previously copied to coprocessor memory, or NFS mounts a remote file system.

**Root file system is a RAM disk image**

If the root is set to be a ram file system, `init` creates a `tmpfs` (Linux* ram disk file system type) in the coprocessor’s memory. It then copies all the files from the initial ram disk image into the new tmpfs mount.

If any RPM files exist in the `/RPMS-to-install` directory, they will be installed. After installation, this directory is removed to free disk space.

The ram disk image is activated as the root device by calling the Linux* `switch_root` utility. This command 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` then performs the normal Linux* user level initialization.

**Root file system is an NFS export**

If an NFS mounted root file system is indicated, the `init` program initializes the `mic0` virtual network interface to the IP address supplied on the kernel command line and mount the NFS export from the host.
As in the ram disk image, the NFS mount 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 NFS mount directory, release all file system memory references to the old initial ram disk and start executing the new `/sbin/init`.

`/sbin/init` performs the normal Linux* user level initialization. All the information required must already be in the NFS export.

### 4.1.5.3 Notifying the host that the coprocessor status is ready

The last step is to notify the host that the coprocessor is ready for access. This causes an interrupt into the host driver which, in turn, updates the coprocessor's state to `online`.

### 4.1.5.4 Coprocessor shutdown

The `mpssd` daemon writes `reset` or `shutdown` to the `/sys/class/mic/micN/state` sysfs node requesting a reset or orderly shutdown of a coprocessor. The `mic.ko` driver, in turn, implements the request operation.

### 4.2 Manual configuration and control

This section describes, at a high level, the considerations and steps in configuring and booting the coprocessor without using the `micctrl` tool or `mpssd` daemon.

In general, this requires:

- Editing configuration files in the default file system image as needed. Typical areas that require attention are networking and user access, the same as for assisted configuration.

- Adding additional software to the coprocessor file system.

- Constructing a coprocessor boot command line.

- Initiating the coprocessor boot and shutdown processes by directly interacting with the `mic.ko` driver.

The default installation automatically loads the `mic.ko` kernel module and starts the `mpss/ofed-mic` services. If this behavior is not desired, switch off the services and remove `/etc/sysconfig/modules/mic.modules`:

```bash
[host]# chkconfig --del ofed-mic
[host]# chkconfig --del mpss
[host]# rm /etc/sysconfig/modules/mic.modules
```

### 4.2.1 Directly editing (and persisting) coprocessor `/etc` files

As described in Section 4.1, assisted configuration of the coprocessor file system is based on overlaying the default file system with a collection of overlay file sets. In that case, the default file system image that is installed as part of Intel® MPSS installation is not modified by the assisted configuration process.
While a similar overlay process could be employed as part of manual configuration, we will assume that the user directly edits the installed default file system.

The default file system image is a compressed CPIO archive, and is installed at `/usr/share/mpss/boot/initramfs-knightscorner.cpio.gz`. To edit files, extract them from the archive:

```
[host]$ gunzip -c /usr/share/mpss/boot/ \ 
initramfs-knightscorner.cpio.gz | cpio -ivd
```

If the file system is to be NFS mounted, it is left in this format. Otherwise, it should be re-archived and compressed for uploading to coprocessor memory:

```
[host]$ find . | cpio -o -H newc | gzip > <ramfs_location>
```

### 4.2.1.1 /init

The default file system’s `/init` script was briefly mentioned in Section 4.1.5.2, and is similar to standard Linux* `/init` scripts. `/init` parses the command line parameters passed to it by the kernel, and performs the following major steps:

- Creates and configures `/etc/modules` and `/etc/modprobe.d/modprobe.conf`.
- Depending on command line parameters, mounts the file system image that the `mic.ko` pushed to coprocessor memory as tmpfs, or NFS mounts a remote export specified in the command line.
- Optionally installs RPM packages that it finds in a special `/RPMS_to_install` directory in the file system image.
- Finally, `/init` switches the root file system to the newly mounted file system image.

If `/init` is edited, for example, to support additional command line options, those changes will need to be propagated to any new version of `/init` in subsequent versions of Intel® MPSS.

### 4.2.1.2 Network configuration and user authentication

Network configuration and user authentication are the most significant configuration tasks, particularly for cluster administration. These topics are treated in detail in Sections 5 and 6 respectively.

### 4.2.1.3 Adding software to the coprocessor’s file system

Although one way to add software is to add files to the file system image, users will generally wish to install RPM-based packages. A simple way to do this is to create an `/RPMS-to-install` directory in the file system image, and place packages to be installed in that directory. The `/init` script, described above, will `rpm install` any packages that it finds in the directory as the last step before performing `switch_root`.

See Section 7 for more information on this topic.
4.2.2 NFS mounting the root and other file systems

/init will NFS mount a remote export if the command line includes the root=nfs command. This command has the syntax:

\[ \text{root=nfs:<server>:<export>} \]

where <server> is the IP address of the exporting node and <export> is the exported directory. For example, the command:

\[ \text{root=nfs:172.31.1.254:/var/mpss/mic0.export} \]

will cause the directory at /var/mpss/mic0.export on node 172.31.1.254 (the default static pair host IP address) to be NFS mounted as root.

The file system to be NFS mounted as root, as well as any other file systems to be NFS mounted, must be described in the /etc/exports file of the exporting host. For example, assume the coprocessor virtual endpoint IP address is 172.31.1.1. To export the host directory /var/mpss/mic0.export, add a descriptor to the host's /etc/exports such as:

\[ /var/mpss/mic0.export 172.31.1.1 (rw,async,no_root_squash) \]

Next call exportfs to update the NFS export tables:

\[ [\text{host}]\# \text{exportfs -a} \]

NFS mounting file systems other than root is done as on any standard Linux* systems. The file system to be exported is described in /etc/exports as shown above, and the mount point is described in the coprocessor's /etc/fstab file. The NFS mounted root file system mount point does not need to be explicitly added to the coprocessor's /etc/fstab because /init mounts it.

For example, assume the host IP address is 172.31.1.254. To mount another host directory /var/mpss/usr.export as /usr on the coprocessor, add a descriptor to the coprocessor's /etc/fstab, for example:

\[ 172.31.1.254:/var/mpss/usr.export /usr nfs defaults 1 1 \]

The mount point, in this case /usr, must exist in the coprocessor file system.

After the coprocessor is rebooted, the remote file system(s) will be mounted onto the coprocessor's files system.

The standard mount command can also be called interactively while the user is logged onto a coprocessor to mount an exported file system.

4.2.3 Driver sysfs settings

The mic.ko driver exports information about installed coprocessors via /sys/class/mic. As described in Section 4.1.5 and below, /sys/class/mic/micN/cmdline and /sys/class/mic/micN/state entries are also used in booting and controlling coprocessors. Appendix C describes these sysfs entries.
4.2.4 Coprocessor-side kernel commandline parameters

As mentioned in Section 4.1.5, a partial command line is written to the mic.ko driver sysfs node `/sys/class/mic/micN/cmdline` at boot time. The driver will augment that command line with additional commands. For example, in assisted configuration, the `mpssd` writes a command line similar to:

```
quiet root=ramfs console=hvc0 cgroup_disable=memory highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_off
```

and a typical augmented command line is:

```
card=0 vnet=dma scif_id=1 scif_addr=0x835c6cd540
vnet_addr=0x831a428118 vcons_hdr_addr=0x831a727540
virtio_addr=0x835c35a9c0 mem=8192M ramoops_size=16384
ramoops_addr=0x8669284000 p2p=1 p2p_proxy=1 etc_comp=1499
reg_cache=1 ulimit=0 huge_page=1 crashkernel=1M@80M quiet
root=ramfs console=hvc0 cgroup_disable=memory highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_off
```

The augmented command line can be read at `/sys/class/mic/micN/kernel_cmdline`.

The mic.ko driver expanded the original kernel command line. The entries `card`, `vnet`, `scif_id`, `scif_addr`, `vnet_addr`, `vcons_hdr_addr`, `virtio_addr`, `mem`, `ramoops_size`, `ramoops_addr`, and `crashkernel` are automatically generated by the driver. These options are non-configurable.

Section 9 describes a range of configuration options, many of which are conveyed to the coprocessor as kernel command line parameters.

4.2.5 Controlling the coprocessor

This section describes how to boot a coprocessor manually, not using micctrl. The mic.ko driver must be loaded:

```
[host]# modprobe mic
```

It is not necessary to start the mpss service (mpssd daemon).

Controlling a coprocessor is then done through the `/sys/class/mic/micN/state` sysfs node. When the state node:

```
[host]# cat /sys/class/mic/micN/state
```

is read, one of the following state values is reported:

- **ready** coprocessor is ready for a boot command
- **booting** coprocessor is currently booting
- **no response** coprocessor is not responding
- **boot failed** coprocessor failed to boot
- **online** coprocessor is currently booted
- **shutdown** coprocessor is currently shutting down
In order to boot or reboot the coprocessor, it must first be in the ready state. If it is in the online state from a previous boot, it can be shut down by writing shutdown to the state node:

```
[host]# echo shutdown > /sys/class/mic/micN/state
[host]$ cat /sys/class/mic/micN/state
shutdown
```

It can be reset by writing reset to the state node:

```
[host]# echo reset > /sys/class/mic/micN/state
[host]$ cat /sys/class/mic/micN/state
resetting
```

Shutting down the coprocessor rather than resetting it is generally recommended particularly if there might be I/O data that must be flushed to some external device.

Both shutdown and reset may take several seconds, so the user must continue to check the state until the coprocessor is reported to be ready:

```
[host]$ cat /sys/class/mic/micN/state
ready
```

Submit the command line:

```
[host]# echo "quiet root=ramfs console=hvc0
cgroup_disable=memory highres=off
micpm=cpufreq_on;corec6_off;pc3_on;pc6_off " > \n/sys/class/mic/micN/cmdline
```

Now boot the coprocessor. For example:

```
[host]# echo "boot:linux:/usr/share/mpss/boot/ \bzimage-knightscorner:/var/mpss/mic0.image.gz" > \n/sys/class/mic/mic0/state
[host]$ cat /sys/class/mic/mic0/state
booting
```

Wait until it is out of the booting state and in the online state:

```
[host]$ cat /sys/class/mic/micN/state
Online
```

The coprocessor is now ready for use. For example you can ssh to it:

```
[host]$ ssh micN
[micN]$ dmesg | tail -n 5
[ 9.529093] blcr: Supports kernel interface version 0.10.3.
```
[ 9.600401] MPSSBOOT Boot acknowledged
[ 17.830104] mic0: no IPv6 routers present
5 Networking configuration

The Intel® Xeon Phi™ coprocessor does not have a hardware Ethernet capability. Instead Virtual Ethernet drivers on the host and coprocessors emulate Ethernet devices to enable the standard TCP/UDP IP stack on the coprocessor. This section describes configuring these endpoints and the construction of networks. Finally, configuration of IP networking over InfiniBand* is discussed.

Assisted and manual networking configurations are addressed separately.

5.1 Assisted configuration

The micctrl utility supports static pair, internal bridge and external bridge topologies. These were briefly described in Section 2.2.3. Using a combination of the Bridge and Network configuration parameters allows to create a diverse and robust network setup.

Each Linux* system in a network uses a host name to identify itself. The Hostname Intel® MPSS configuration parameter is used to configure the host name of the coprocessor.

Each network interface is identified by its MAC address. Virtual network endpoints on the host and on a coprocessor require their own unique address. These addresses are configured using the MacAddrs parameter.

For the purpose of network configuration, several files are added or modified, based on the host OS type (Red Hat* or SUSE*). These may include:

```
/etc/hosts
/etc/network/interfaces # SUSE*
/etc/sysconfig/network-scripts/ifcfg-*; # RHEL*: various depending on network topology
```

On the coprocessor file systems the files added are:

```
/etc/network/interfaces
/etc/hostname
/etc/ssh/ssh_host_key
/etc/ssh/ssh_host_key.pub
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_rsa_key.pub
/etc/ssh/ssh_host_dsa_key
/etc/ssh/ssh_host_dsa_key.pub
/etc/ssh/ssh_host_ecdsa_key # if present
/etc/ssh/ssh_host_ecdsa_key.pub # if present
/etc/resolv.conf
/etc/nsswitch.conf
/etc/hosts
```

All network configuration parameters take effect upon executing `service mpss start`. 
5.1.1 Host SSH keys

The secure shell utilities recognize a Linux* system on the network by its "host key files". These files are found in the /etc/ssh directory. The host key values, like the MAC addresses, are considered to be highly persistent, and the micctrl command will retain their values provided they exist.

In some clusters, detecting and protecting against "man in the middle" and other such attacks might not be required. In this case, the system administrator may use the micctrl --hostkeys command to set the host SSH keys to be the same cluster wide.

5.1.2 Name resolution configuration

micctrl --initdefaults configures name resolution on the coprocessors by creating an /etc/nsswitch.conf file and copying the /etc/resolv.conf file from the host to the coprocessor's file systems.

5.1.3 Host name assignment

Each coprocessor needs its own host name. The Hostname parameter in each micN.conf configuration file defines the host name of the corresponding coprocessor. Parameter syntax is:

```
Hostname <name>
```

The default value set by the micctrl --initdefaults command is:

```
Hostname <short_host_name>-micN.<domain>
```

where `<short_host_name>` is the name returned by:

```
[host]$ hostname --short
```

`<domain>` is the host's domain name. For example, if the host's hostname is `abc.xyz.com`, then the coprocessor hostname will be `abc-micN.xyz.com`. The host name string may be changed by editing the `micN.conf` configuration file.

5.1.4 MAC address assignment

Because the coprocessor does not have a hardware network interface, its network endpoint does not have a pre-assigned MAC address. Therefore a MAC address must be generated and assigned to each virtual network device; several options are available to facilitate this operation.

At driver load time, the host and coprocessor drivers generate MAC addresses for their respective endpoints, setting the first three octets to 4C:79:BA. This occurs regardless of whether configuration is assisted or manual.

Normally, these MAC addresses are based on the coprocessor’s serial number and are consistent across restarts. Some early coprocessors lacked serial numbers; MAC addresses of those devices are generated randomly.

It is recommended to use the default serial number based MAC addresses, however, these can be overridden if necessary.
MAC assignment is controlled by the `MacAddrs` configuration parameter in the `micN.conf` configuration file:

```
MacAddrs Serial|Random|<host MAC>:<card MAC>
```

The initial parameter created by `micctrl --initdefaults` is:

```
MacAddrs Serial
```

This specifies serial number-based MAC address generation. In addition to random MAC address generation, explicit host and coprocessor can be assigned. See Appendix A.5.2 for details.

The `micctrl --mac` command:

```
micctrl --mac=serial|random|<MAC address>
```

can be used to modify the MacAddrs parameter in one or more `micN.conf` configuration files. This parameter can also be edited manually. See Appendix B.4.5.1 for details.

5.2 Network topologies

This section describes configuration of each of the basic network topologies.

*Note:* The `mpss` service must be stopped before using `micctrl` to configure the network topology:

```
[host]# service mpss stop
```

5.2.1 Static pair configuration

In the static pair topology, each coprocessor is assigned to a separate subnet known only to the host. Only static IP address assignment is supported. The `Network` configuration parameter format for static pair networking is described in detail in Appendix A.5.3.

5.2.1.1 Static pair configuration using micctrl

Static pair network topology can be partially configured by directly editing the `Network` configuration parameter. To complete the configuration additional steps are also required. Therefore the recommended method of changing the network configuration is to use the `micctrl --network` command. Specifically, this command will edit configuration files to remove the current network configuration before implementing the new one. The `micctrl --network` command also creates and/or modifies host and coprocessor network configuration files, and brings network endpoints on the host down and up as needed.

Configuring a static pair network using the `micctrl --network` command is described in detail in Appendix B.4.5.3.
5.2.1.2 Implementing the static pair configuration

This section describes in some detail the edits and other operations that micctrl performs when the micctrl --network command is used to configure a static pair network topology. The information in this section is not required in order to use these micctrl commands. The reader can skip this section unless a deeper understanding of the configuration process is needed.

For explanatory purposes we will assume the following command is executed on a host system with two coprocessors:

```
[host]# micctrl --network=static --ip=172.31
```

In this case, micctrl will set the third quad of each IP address to N+1 for each coprocessor with a name specified to micN. The fourth quad of the host endpoint IP address will be 254, and the coprocessor endpoint IP address will be 1. MTU will default to 64512, and modhost and modcard will both default to yes (see Appendix B.3.2.8 and B.3.2.9 for details on modhost and modcard suboptions to micctrl).

micctrl first parses the Network configuration parameter in each of the /etc/mpss/micN.conf files to determine the existing network configuration. Next it shuts down the current virtual network connections using the ifdown micN command for each of the coprocessors, deletes existing /etc/sysconfig/network-scripts/ifcfg-micN files, removes the micN entries from /etc/hosts, and then creates a new /etc/sysconfig/network-scripts/ifcfg-micN file for each coprocessor.

The ifcfg-mic0 will now have contents similar to:

```
DEVICE="mic0"
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=172.31.1.254
NETMASK=255.255.255.0
MTU=64512
```

In general, an identical /etc/sysconfig/network-scripts/ifcfg-micN file is created for each micN with DEVICE=micN and IPADDR=172.31.1+N.254.

micctrl now executes ifup micN for each of the coprocessors. At this time, the ifconfig command relevant output should be similar to:

```
mic0    Link encap:Ethernet
inet addr:172.31.1.254  Bcast:172.31.1.255  Mask:255.255.255.0
mic1    Link encap:Ethernet
inet addr:172.31.2.254  Bcast:172.31.2.255  Mask:255.255.255.0
```

showing that the two host endpoints have the IP address specified by the micctrl --network command.

micctrl then creates/updates the network configuration files for the coprocessor's file system. It will first create/update the network interface configuration file /var/mpss-/mic0/etc/network/interfaces with the contents:
Networking configuration

# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
auto mic0
iface mic0 inet static
   address 172.31.1.1
gateway 172.31.1.254
   netmask 255.255.255.0
   mtu 64512

The /var/mpss/mic1/etc/network/interfaces file is similar.

Next, micctrl --network replaces the Network configuration parameter in each coprocessor's configuration file with a new parameter. For example the /etc/mpss/mic0.conf file will now include the Network configuration parameter:

```
Network class=StaticPair micip=172.31.1.1 hostip=172.31.1.254
modhost=yes modcard=yes netbits=24 mtu=64512
```

micctrl now updates the /etc/hosts file to include descriptors of the remote endpoints:

```
172.31.1.1  blutune-mic0.music.local mic0 #Generated-by-micctrl
172.31.1.2  blutune-mic1.music.local mic1 #Generated-by-micctrl
```

and then creates/upates the /var/mpss/micN/etc/hosts files to have content similar to the following (/var/mpss/mic0/etc/hosts shown):

```
127.0.0.1   localhost.localdomain localhost
::1         localhost.localdomain localhost
172.31.1.254 host blutune.music.local
172.31.1.1   mic0 blutune-mic0.music.local mic0
172.31.2.1   blutune-mic1.music.local mic1
```

Coprocessors will use the new configuration after the next boot.

5.2.2 Internal bridge configuration

Linux* provides a mechanism for bridging network devices to a common network. The term internal bridge, in the context of the coprocessor configuration, refers to a network of multiple coprocessor virtual network endpoints that are connected through a host bridge endpoint. Only static IP address assignment is supported.

This network topology depends on a Bridge parameter in the default.conf configuration file and a Network parameter in micN.conf configuration file of each coprocessor to be included in the bridge. The Bridge and Network parameters for the internal bridge configuration are described in detail in Appendix A.5.4.
**5.2.2.1 Internal bridge configuration using micctrl**

Although the internal bridge network can be partially configured by directly editing the Bridge and Network configuration parameters, other steps are also required. Therefore the recommended method of changing the network configuration is to use the micctrl --addbridge and micctrl --network command. This command will edit the configuration files to remove the current network configuration before implementing the new one. micctrl --network also creates and/or modifies host and coprocessor network configuration files, and brings network endpoints on the host up and down as needed.

Configuring an internal bridge network using the micctrl --addbridge and micctrl --network command is described in detail in Appendix B.4.5.4.

**5.2.2.2 Implementing the internal bridge configuration**

This section describes in some detail the modifications and other actions performed when the micctrl --network and --addbridge commands are used to configure an internal bridge network topology. The information in this section is not required in order to use these micctrl commands. Reader may choose to skip this section unless a deeper understanding of the configuration process is needed.

For explanatory purposes we will assume the following commands are executed on a host system with two coprocessors:

```
[host]# micctrl --addbridge=br0 --type=internal --ip=172.31.1.254
[host]# micctrl --network=static --bridge=br0 --ip=172.31.1.1
```

The micctrl --addbridge command performs a series of steps starting with removing the current network configuration. micctrl first parses the Network configuration parameter in each of the /etc/mps/micN.conf files and the Bridge parameter in the /etc/mpss/default.conf file to determine the existing network configuration.

micctrl then adds/modifies the Bridge parameter in the /etc/mpss/default.conf file to contain:

```
Bridge br0 Internal 172.31.1.254 24 64512
```

The value 24 in this parameter is the default netbits value, defining a netmask of FFFFFFFF00. The value 64512 is the default MTU value.

Then, the host configuration file, /etc/sysconfig/network-scripts/ifcfg-br0, is created or modified to describe the bridge with contents similar to:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=172.31.1.254
NETMASK=255.255.255.0
```

The micctrl utility then executes the ifup br0 command to bring up the bridge interface.
The `micctrl --network` command slaves the host ends of the virtual networks to the designated bridge `br0`, and replaces the network configuration files for the coprocessors with a configuration for the new IP addresses. `micctrl` again parses the `Network configuration parameter in each of the /etc/mpss/micN.conf` files to determine the existing network configuration.

`micctrl` next shuts down the current virtual network connections using the `ifdown micN` command for each of the coprocessors, deletes existing `/etc/sysconfig/network-scripts/ifcfg-micN` files, removes the `micN` entries from `/etc/hosts`, and then creates a new `/etc/sysconfig/network-scripts/ifcfg-micN` file for each coprocessor. The `ifcfg-mic0` will now have contents similar to:

```
DEVICE=mic0
ONBOOT=yes
NM_CONTROLLED="no"
BRIDGE=br0
MTU=64512
```

where `BRIDGE=br0` causes the new endpoint to be added to the bridge. In general, identical `/etc/sysconfig/network-scripts/ifcfg-micN` files are created for each micN with `DEVICE=micN`.

When this operation completes, `micctrl` executes `ifup micN`, for each coprocessor. At the end of this process, the `brctl` show command can be used to check the status of the bridge. Its output should be similar to:

```
bridge name bridge id  STP enabled interfaces
br0  8000.66a8476a8f15  no  mic0
     8000.66a8476a8f15  no  mic1
```

The `ifconfig` command relevant output should be:

```
br0 Link encap:Ethernet
     inet addr: 172.31.1.254  Bcast: 172.31.1.255
     Mask:255.255.255.0
mic0 Link encap:Ethernet
mic1 Link encap:Ethernet
```

These commands show that the mic0 and mic1 virtual network interfaces are slaved to bridge br0. Bridge br0 has been assigned the IP address specified by the `micctrl --addbridge` command, while the slaves do not have their host IP addresses.

Additionally `micctrl` creates the network configuration files for the coprocessors' file system. It will first create/update the network interface configuration file `/var/mpss/mic0/etc/network/interfaces` with the contents:

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
```

auto mic0
iface mic0 inet static
    address 172.31.1.1
gateway 172.31.1.254
    netmask 255.255.255.0

The /var/mpss/mic1/etc/network/interfaces file is similar.

The existing Network configuration parameter in each coprocessor’s configuration file is then replaced with a new parameter. For example the /etc/mpss/mic0.conf file now has the Network configuration line:

    Network class=StaticBridge bridge=br0 micip=172.31.1.1
    modhost=yes modcard=yes

The /etc/mpss/mic1.conf file will have the same line with the exception that the IP address is 172.31.1.2.

micctrl now updates the /etc/hosts file to include descriptors of the remote endpoints:

    172.31.1.1 blutune-mic0.music.local mic0 #Generated-by-micctrl
    172.31.1.2 blutune-mic1.music.local mic1 #Generated-by-micctrl

and then creates/updates the /var/mpss/micN/etc/hosts files to have content similar to the following (/var/mpss/mic0/etc/hosts shown):

    127.0.0.1 localhost.localdomain localhost
    ::1 localhost.localdomain localhost
    172.31.1.254 host blutune.music.local
    172.31.1.1 mic0 blutune-mic1.music.local mic0
    172.31.1.2 mic1 blutune-mic1.music.local mic1

In general, each coprocessor's /etc/hosts file includes the IP addresses and host names of all coprocessors on the internal bridge network.

Coprocessors will use the new configuration after the next boot.

5.2.3 External bridge configuration

The Linux* bridging mechanism can also bridge the coprocessor virtual connections to a physical Ethernet device. In this topology, the virtual network interfaces become configurable to the wider subnet. Both static IP address assignment and DHCP based IP address assignment/reservation are supported.

This network topology depends on a Bridge in the default.conf configuration file and a Network parameter micN.conf configuration file of each coprocessor to be included in the bridge. The Bridge and Network parameters for the external bridge configuration are described in detail in Appendix A.5.5.

5.2.3.1 External bridge configuration using micctrl

External bridge network can be partially configured by editing the Bridge and Network configuration parameters directly, however, other steps are also required. The recommended method of changing the network configuration is to use the
micctrl --addbridge and micctrl --network command. Specifically, those commands will edit configuration files as needed to remove the current network configuration before implementing new one. micctrl --network also creates and/or modifies host and coprocessor network configuration files, and brings network endpoints on the host up and down as needed.

Configuring an external bridge network using the micctrl --addbridge and micctrl --network command is described in detail in Appendix B.4.5.5.

5.2.3.2 Implementing the external bridge configuration

This section describes in some detail the edits and other operations that micctrl performs when the micctrl --network and --addbridge commands are used to configure an external bridge network topology. The information in this section is not required in order to use these micctrl commands. Readers may skip this section unless a deeper understanding of the configuration process is needed.

When IP address assignment is static, micctrl performs the same steps as for the Internal Bridge configuration, except that the default MTU size is set to 1500 bytes.

For dhcp based IP address assignment, the steps are similar except that the bridge descriptor file, for example /etc/sysconfig/network-scripts/ifcfg-br0, will specify dhcp address assignment. For example:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=dhcp
NETMASK=255.255.255.0
MTU=1500
```

BOOTPROTO is set to dhcp rather than static, and there is no IPADDR parameter. Similarly, each coprocessor endpoint must be described in that coprocessor’s /var/mpss/micN/etc/network/interfaces file with contents similar to:

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
auto mic0
iface mic0 inet dhcp
    pre-up /bin/ip link set $IFACE mtu 1500
    hostname bjhondo-desktop7-mic0.dd.domain.com
```

This configures the micN coprocessor endpoint for DHCP IP address assignment and configures the endpoint mtu to 1500 bytes for compatibility with other devices.
Because IP addresses are assigned by the dhcp server, the host and coprocessor /etc/hosts files are not modified.

5.3 Manual configuration

Manual network configuration mostly requires editing standard configuration files on the host and on the coprocessor file systems. Generally speaking, this includes the host and coprocessor configuration files listed in Section 4.1.1.2. To edit or add files to the default file system image, refer to Section 4.2.1.

**Note:** Network configuration on the coprocessor is Debian* based. In particular, a single /etc/network/interfaces file describes all endpoints. Because each coprocessor has only a single network endpoint, this file is generally quite simple.

The default file system image, as installed, already includes several of these files, specifically:

```
/etc/network/interfaces
/etc/hostname
/etc/nsswitch.conf
/etc/hosts
```

Each of these must be modified to complete network configuration.

5.3.1 Host name

The /etc/hostname file in the coprocessor’s file system image should contain the coprocessor host name.

5.3.2 Mac addresses

For manual configuration, nothing needs to be done provided that serial number-based MAC address generation is acceptable.

To assign an explicit MAC address to a coprocessor edit the /etc/network/interfaces file in its file system image by adding the following line in the section describing the micN endpoint:

```
hwaddress ether XX:XX:XX:XX:XX:XX
```

Standard Linux* utilities such as *ifconfig* can also be used to change the MAC address of host endpoints. For example:

```
[host]# ifconfig mic0 hw ether 4A:79:BA:15:00:21
```

This will set the mic0 host endpoint MAC address to 4A:79:BA:15:00:21. However, this direct assignment is not persistent. When the host driver is restarted, the MAC address will revert to the default value.
5.3.3 Network topologies

This section describes in some detail the edits and other operations to manually configure each of the basic network topologies. Because IP address assignment is an intrinsic part of the network configuration, it is described in the following sections.

For the purpose of this guide it assumed that a platform with two coprocessors is established, and that virtual network endpoints are given micN names, for instance mic0 for coprocessor 0. We also assume that the coprocessors have been reset and are in the ready state, and that previous network endpoints and bridges have been shut down, for example, by using the ifdown command.

5.3.3.1 Static pair

To define the host endpoint of each static pair, create and/or edit the /etc/sysconfig/network-scripts/ifcfg-micN file for each coprocessor to be paired, and assign the chosen device name, IP address, netmask, and MTU value. Resulting ifcfg-mic0 file should have content similar to the following example:

```
DEVICE="mic0"
TYPE=Ethernet
ONBOOT=yes
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=172.31.1.254
NETMASK=255.255.255.0
MTU=64512
```

In general, an identical /etc/sysconfig/network-scripts/ifcfg-micN file is created for each micN with DEVICE=micN and IPADDR=172.31.1+N.254.

Each coprocessor endpoint must be described in that coprocessor’s /network/interfaces file with contents similar to:

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
auto mic0
iface mic0 inet static
  address 172.31.1.1
  gateway 172.31.1.254
  netmask 255.255.255.0
  mtu 64512
```

The host and coprocessor IP addresses must be from the same subnet.

A descriptor of each coprocessor endpoint should be added to the host’s /etc/hosts file to associate IP addresses with the coprocessor hostnames. For example:

```
```
Similarly, a descriptor of the corresponding host endpoint should be added to each coprocessor’s `/etc/hosts` file to associate the host’s endpoint IP address with the host’s hostnames. For example, mic0’s `/etc/hosts` might contain:

```
127.0.0.1   localhost.localdomain localhost
::1         localhost.localdomain localhost
172.31.1.254 host blutune.music.local
172.31.1.1   mic0 blutune-mic0.music.local mic0
172.31.2.254 blutune-mic0.music.local mic0
172.31.2.1   blutune-mic1.music.local mic1
```

For this example, `/etc/hosts` includes descriptors of both the host endpoint and the local endpoint.

Each of these endpoints can now be brought up by calling the `ifup micN` command for each bridged coprocessor. At this point the `ifconfig` command relevant output should be similar to:

```
mic0    Link encap:Ethernet
       inet addr:172.31.1.254  Bcast:172.31.1.255  Mask:255.255.255.0
mic1    Link encap:Ethernet
       inet addr:172.31.2.254  Bcast:172.31.2.255  Mask:255.255.255.0
```

The coprocessors will use the new network configuration after the next boot.

### 5.3.3.2 Internal bridge

To define the host bridge endpoint, create and/or edit a standard interface configuration file with the chosen bridge name, for example `/etc/sysconfig/network-scripts/ifcfg-br0`, assigning the chosen device name, IP address, netmask, and mtu value. For example, `ifcfg-br0` should have content similar to:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=172.31.1.254
NETMASK=255.255.255.0
```

A standard host interface file, `/etc/sysconfig/network-scripts/ifcfg-micN`, must be created for each coprocessor that is to be slaved to the bridge. File contents should be similar to:

```
DEVICE=mic0
ONBOOT=yes
NM_CONTROLLED="no"
BRIDGE=br0
MTU=64512
```
**Note:** Bridged host endpoints do not have IP addresses.

Each coprocessor endpoint must be described in that coprocessor’s `/var/mpss/micN/etc-/network/interfaces` file with contents similar to:

```bash
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
auto mic0
iface mic0 inet static
   address 172.31.1.1
   gateway 172.31.1.254
   netmask 255.255.255.0
```

The bridge and coprocessor IP addresses must be from the same subnet.

The host’s `/etc/hosts` file must contain a descriptor of coprocessor endpoint to associate IP addresses with the coprocessor hostnames. For example:

```plaintext
172.31.1.1  blutune-mic0.music.local  mic0
172.31.1.2  blutune-mic1.music.local  mic1
```

Similarly, a descriptor of the corresponding host bridge endpoint should be added to each coprocessor’s `/etc/hosts` file to associate the host’s endpoint IP address with the host’s hostnames. For example, `mic0`’s `/etc/hosts` might contain:

```plaintext
127.0.0.1   localhost.localdomain  localhost
::1        localhost.localdomain  localhost
172.31.1.254 host  blutune.music.local
172.31.1.1  mic0  blutune-mic1.music.local  mic0
172.31.1.2  mic1  blutune-mic1.music.local  mic1
```

In this example `/etc/hosts` includes descriptors of the local endpoint, the host endpoint, and the other coprocessors.

The bridge interface can now be brought up using the `ifup br0` command, and each host endpoint can now be brought up by calling the `ifup micN` command for each bridged coprocessor. At this point the `brctl show` command can be used to check the status of the bridge. Its output should be similar to:

```plaintext
bridge name  bridge id  STP enabled  interfaces
br0  8000.66a8476a8f15  no  mic0
     mic1
```

The `ifconfig` command relevant output should be similar to:

```plaintext
br0  Link encap:Ethernet
     inet addr: 172.31.1.254  Bcast: 172.31.1.255
     Mask: 255.255.255.0
```
These commands show that the mic0 and mic1 virtual network interfaces are slaved to bridge br0.

Coprocessors will use the new configuration after the next boot.

### 5.3.3.3 External bridge

The external bridge configuration requires slaving the physical Ethernet endpoint to the bridge.

When IP address assignment is static, configuration is the same as for the internal bridge, except that the default mtu size should be changed to 1500 bytes.

If DHCP based IP address assignment is dynamic, the steps are similar except that the bridge descriptor file, for example, `/etc/sysconfig/network-scripts/ifcfg-br0`, will be similar to:

```bash
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=dhcp
NETMASK=255.255.255.0
MTU=1500
```

with `BOOTPROTO` now set to `dhcp` rather than `static`, and with no `IPADDR` parameter.

In both the static and dynamic IP address assignment cases, it is the system administrator’s responsibility to add the gateway to the host network bridge configuration. For example add `GATEWAY=10.23.185.1` to `/etc/sysconfig/network-scripts/ifcfg-br0`.

Similarly, each coprocessor endpoint must be described in that coprocessor’s `/var/mpss/micN/etc/network/interfaces` file with contents similar to:

```bash
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# MIC virtual interface
auto mic0
iface mic0 inet dhcp
    pre-up /bin/ip link set $IFACE mtu 1500
    hostname whsniddo-desktop8-mic0.dd.domain.com

```

This configures the mic0 coprocessor endpoint for DHCP IP address assignment and configures the endpoint MTU to 1500 bytes for compatibility with other devices.
5.4 IPoIB networking configuration

The OFED IPoIB driver is an implementation of the IP over InfiniBand protocol as specified by RFC 4391 and 4392, issued by the IETF IPoIB working group. It is a native implementation in the sense of setting the interface type to ARPHRD_INFINIBAND and the hardware address length to 20 versus implementations that are masqueraded to the kernel as Ethernet interfaces.

The module runs on top of coprocessor’s CCL-Direct Kernel IB Verbs. As a result, most of the functional and performance characteristics are bound by CCL-Direct restrictions. The driver is released to enable InfiniBand*-based Lustre* solutions that require IPoIB interface regardless of LNET configurations.

Figure 13: One-to-One IB Device (HCA, Port) Mapping

5.4.1 Managing the IPoIB interface

The coprocessor’s IPoIB currently manages the virtual IB devices via CCL-Direct IBP proxy drivers. Its existing configuration parameters are inherited from OFED settings without change.

To enable the IPoIB interface on the coprocessor from the host, edit the /etc/mpss/ipoib.conf file to bring up the ib0 interface on a coprocessor with the default hostname (mic0):

```plaintext
ipoib_enabled=yes
mic0_ib0=192.168.100.100
```
5.4.2 IP addressing

Unlike the coprocessor’s Ethernet virtual driver, IPoIB does not require bridging or routing to be configured. In the default case, there is an automatically created one-to-one mapping of the (HCA, Port) pairs between the host and coprocessor. Figure 13 shows an example configuration with two 2-port HCAs on the host. All 8 ports (host and coprocessor combined) can be individually configured by net-if commands. On the coprocessor node, the setting is configured by ifconfig command, by adding a configuration file in /etc/sysconfig/network, or by editing the /etc/mpss/ipoib.conf file. The host side follows the host OS conventions.

5.4.3 Datagram vs. connected modes

The driver supports two modes of operation: datagram and connected. The mode is set and read through the interface’s /sys/class/net/<intf name>/mode file. Datagram is the default mode.

In the datagram mode, the CCL-Direct IB UD transport is used, and the IPoIB MTU is equal to the IB L2 MTU minus the IPoIB encapsulation header (4 bytes). For example, in a typical IB fabric with a 2K MTU, the IPoIB MTU will be 2048 - 4 = 2044 bytes.

In the connected mode, the IB RC transport is used. Connected mode takes advantage of the connected nature of the IB transport that allows an MTU up to the maximal IP packet size of 64K. This reduces the number of IP packets needed for handling large UDP datagrams and TCP segments, and increases the performance for large messages.
6 User credentialing and authentication

The coprocessor’s Linux* operating system supports SSH access using SSH keys and/or password authentication, requiring that valid credentials are available to the coprocessor’s OS. In addition, some offload options require that specific user credentials be configured on the coprocessor. Refer to the discussion on COI Authorized user ownership in Section 9.4.1.1.2 for details.

The coprocessor’s OS obtains user credentials from standard configuration files such as /etc/passwd and /etc/shadow in the coprocessor’s file system, or from an LDAP or NIS server. The OS looks for users’ SSH keys in their $HOME/.ssh directories.

micctrl can be used to populate /etc/passwd, /etc/shadow, and SSH key files in the coprocessor’s file system, or those files can be edited directly. In addition, micctrl can be used to configure the coprocessor’s OS to access an LDAP or NIS server for user credentials, or this configuration can be performed by directly editing LDAP or NIS configuration files.

6.1 Assisted configuration of user credentials

Assisted configuration of user credentialing is performed entirely through micctrl operations. There are no parameters in the Intel® MPSS default.conf and micN.conf files that apply.

6.1.1 Local configuration

Several micctrl commands support configuring user credentials. The micctrl --initdefaults command creates and initializes /var/mpss/micN/etc/passwd and /var/mpss/micN/etc-/shadow in the per-coprocessor /var/mpss/micN overlay set of each specified coprocessor, unless those files already exist. The --users and --pass parameters control which user accounts populate those files and whether passwords are copied to the coprocessor. In the event that those files already exist, micctrl --initdefaults will not change them unless the --users and/or --pass parameters require that these files be deleted and recreated with a different set of data. The micctrl command always creates these files if they did not previously exist, and the --users and --pass parameters control how these files are populated.

micctrl --initdefaults also populates /var/mpss/micN/etc/group with the group attributes of each user in /var/mpss/micN/etc/passwd.

When micctrl --initdefaults (re)creates /var/mpss/micN/etc/passwd, for each <user> in /var/mpss/micN/etc/passwd, it also copies the files in /home/<user>/.ssh/ to /var/mpss/micN/home/<user>/.ssh. Similarly it will copy files from /root/.ssh/ to /var/mpss/micN/root/.ssh. The users sshd, nobody, nfsnobody and micuser do not have SSH keys.
The --nocreate parameter to `micctrl --initdefaults` suppresses population of 
/var/mpss/micN/home/<user> directories. This can save ram file system memory when 
LDAP home directory auto mount is enabled or the /home directories are NFS mounted.

Other `micctrl --initdefaults` parameters are unrelated to user credentialing. Refer to 
Appendix B.4.2.1 for additional details on `micctrl --initdefaults`.

`micctrl --initdefaults` is designed to establish an initial user credential configuration. 
Other `micctrl` commands are intended to support adding, modifying, and/or removing 
user credentials as needed. The default user credentialing behavior on the coprocessor 
can be customized with the `micctrl --userupdate` command. This command duplicates 
the semantics of `micctrl --initdefaults` with respect to user credentials and SSH keys. 
Refer to Appendix B.4.6.1 for additional details.

The `micctrl --useradd` command can be used to add a specified users attributes to 
/var/mpss/micN/etc/passwd and /var/mpss/micN/etc/shadow. This command would 
typically be called after a new user is added on the host. If a specified coprocessor is in 
the online (running) state, the corresponding changes are applied dynamically to its file 
system. Refer to Appendix B.4.6.2 for additional details.

The `micctrl --userdel` command removes user credentials, and optionally the user’s 
home directory from the current configuration. Specifically, the user is removed from 
/var/mpss/micN/etc/passwd and /var/mpss/micN/etc/shadow of the specified 
coprocessors, and /var/mpss/micN/home/<user> is optionally deleted. If a specified 
coprocessor is in the online (running) state, the corresponding changes are applied 
dynamically to its file system. Refer to Appendix B.4.6.3 for additional details.

The `micctrl --passwd` command allows a non-privileged user to change their password 
on the host and in the current configuration. Root can use this command to change the 
password of any user. If a specified coprocessor is in the online (running) state, the 
corresponding changes are applied dynamically to its file system. Refer to Appendix B.4.6.4 for additional details.

The `micctrl --groupadd` and `--groupdel` commands enable adding and/or removing a 
specified group from the configuration. If a specified coprocessor is in the online 
(running) state, the corresponding changes are applied dynamically to its file system. Refer to Appendix B.4.6.5 and Appendix B.4.6.6 for additional details.

The `micctrl --hostkeys` command can be used to populate the /var/mpss/micN/etc/ssh directory with some previously created keys. For example, the keys in 
/var/mpss/micN/etc/ssh might be copied, using the standard cp command, to some 
temporary directory before calling `micctrl --cleanconfig` and `micctrl --initdefaults`. Then 
micctrl --hostkeys can be called to restore those keys, overwriting the new host keys 
which `micctrl --initdefaults` generated. By doing this the corresponding micN coprocessor 
will continue to be recognized as a known host. Refer to Appendix B.4.6.7 for additional 
details.

The `micctrl --sshkeys` command copies the *.pub public SSH keys of a user, <user>, to 
/var/mpss/micN/home/<user>/.ssh. This command might be called in the event that a 
user’s ssh keys are created or changed after the initial configuration is established. 
Refer to Appendix B.4.6.8 for additional details.
6.1.2 Enabling the LDAP service

The coprocessor can use the LDAP service for user authentication.

The network must be configured to enable access to the LDAP server, which typically will not be on the local host. Thus, to be able to access the LDAP server from the coprocessor, the external bridge configuration should be used. See Section 5.2.3 for details.

An LDAP client is not preinstalled in the coprocessor default file system and therefore must be added. The `micctrl --rpmdir` command:

```
[host]# micctrl --rpmdir=$MPSS38_K1OM
```

creates a configuration parameter that tells the `micctrl --ldap` command where to find the RPMs that it will need to install the LDAP service, so that it can configure the needed RPM overlay parameters.

The `micctrl --ldap` command:

```
[host]# micctrl --ldap=(<server>|default) --base=<domain> \ [mic card list]
```

is then used to configure the coprocessor OS to use LDAP for user authentication. The `<server>` value specifies the LDAP authentication server to be used, and the `base` argument specifies the domain to be used. For example:

```
[host]# micctrl --ldap=192.168.122.129 --base="example.com"
```

The `--ldap=disable` option disables the LDAP authentication.

6.1.3 Enabling the NIS service

The coprocessor can use the NIS service for user authentication.

Since the NIS server will not be running on the local host, the network must be configured to enable access to a remote NIS server. To be able to access the NIS server from the coprocessor, the external bridge configuration should be used. See Section 5.2.3 for details.

The NIS client is not preinstalled in the coprocessor default file system and, therefore must be installed. The `micctrl --rpmdir` command

```
micctrl --rpmdir=$MPSS38_K1OM
```

creates a configuration parameter that the needed RPMs can be found in the `$MPSS38_K1OM` directory.

The `micctrl --nis` command:

```
micctrl --nis=(<server>|default) --domain=<domain> [mic card list]
```
is then used to configure the coprocessor OS to use NIS for user authentication. The `<server>` value specifies the NIS authentication server to be used, and the `domain` argument specifies the domain to be used. For example:

```
[host]# micctrl --nis=192.168.122.129 --domain="example.com"
```

The `--nis=disable` option disables the NIS authentication.

## 6.2 Manual configuration of user credentials

`micctrl` provides credentialing support that is sufficient for many situations. However, particularly in a cluster environment, configuring services such as LDAP may require cluster-specific configuration. This section briefly discusses basic file-based credentialing and then provides step by step instructions for enabling LDAP, NIS and SSH based authentication. These latter instructions are intended as a starting reference; it is expected that the system administrator may wish to refine or customize these configurations.

### 6.2.1 Configuration file based credentialing

Section 4.2.1 discussed how to directly edit configuration files that will be in a coprocessor’s file system. The same general guidelines apply to creating and editing a coprocessor’s `/etc/passwd`, `/etc/shadow`, `~/.profile`, and files in `~/.ssh` for each user (including root) that is to have access to the coprocessor.

As described in Section 4.2, one must reboot a coprocessor in order for changes to user credentialing to take effect. Alternatively, credentialing can be changed dynamically via SSH.

#### 6.2.1.1 Enabling the LDAP service for credentialing

LDAP service on a coprocessor can be configured manually. The network must be configured to enable access to the LDAP server, which typically will not be on the host. In that case the network should be configured as an external bridge so the LDAP server can be reached from the coprocessor. See Section 5.3.3.3 for details.

The following steps document enabling LDAP service. This particular configuration does not allow changing the user’s password from the coprocessor.

1. Install `nss-ldap` and `pam-ldap` RPM files into the coprocessor file system. These RPMs are included in the `mpss-3.8-k1om.tar` file. There are several ways to install these. See Section 7 to learn about various approaches to adding software. In this example, required RPMs are copied from `$MPSS38_K1OM` to a booted coprocessor micN:

   ```
   [host]$ scp $MPSS38_K1OM/nss-ldap-265-r0.klom.rpm micN:/tmp
   [host]$ scp $MPSS38_K1OM/pam-ldap-186-r0.klom.rpm micN:/tmp
   ```

   and RPM installed:
2. **Configure nss-ldap.** Edit `/etc/nsswitch.conf` to add LDAP to the services you wish to enable, and add the coprocessor host information to `/etc/hosts`.

   ```bash
   [micN]# cp /etc/nsswitch.ldap /etc/nsswitch.conf
   [micN]# sed -ie "s/hosts:/s/dns ldap/files/" /etc/nsswitch.conf
   [micN]# SelfIp=`/sbin/ifconfig mic0 | grep "inet addr" | \cut -d":" -f2 | cut -d" " -f1`
   [micN]# echo ${SelfIp} `hostname` `hostname -s` >> /etc/hosts
   ```

3. **Configure LDAP.** Add the LDAP server and base domain name to `/etc/ldap.conf`.

   ```bash
   [micN]# cp /etc/openldap/ldap.conf /etc
   [micN]# echo "URI ldap://<LDAP server IP address>" >> /etc/ldap.conf
   [micN]# echo "BASE dc=example,dc=com" >> /etc/ldap.conf
   ```

4. **Configure PAM to allow the LDAP module for SSH and others.**

   ```bash
   [micN]# sed -ie "s/^$/auth sufficient pam_ldap.so/" /etc/pam.d/common-auth
   [micN]# sed -ie "/session/s/required/optional/" /etc/pam.d/sshd
   ```

### 6.2.1.2 Enabling the NIS/YP service for credentialing

The NIS service on a coprocessor can be configured manually.

The network must be configured to enable the coprocessor to access the NIS server, which typically will not be on the host. In that case the network should be configured as an external bridge. See Section 5.3.3.3 for details.

The following steps document enabling NIS service. This particular configuration does not allow changing the user's password from the coprocessor.

1. Install `rpcbind`, `ypbind-mt`, `yp-tools`, and `glibc-extra-nss` RPM files on the coprocessor. These RPMs are included in the `mpss-3.8-k1om.tar` file. There are several ways to install these. See Section 7 to learn about other approaches to adding software. In this example, required RPMs are copied from `$MPSS38_K1OM` to a booted coprocessor micN:

   ```bash
   [host]$ scp $MPSS38_K1OM/rpcbind-0.*.k1om.rpm micN:/tmp
   [host]$ scp $MPSS38_K1OM/yp-tools-*.*.k1om.rpm micN:/tmp
   [host]$ scp $MPSS38_K1OM/ypbind-mt-*.*.k1om.rpm micN:/tmp
   [host]$ scp $MPSS38_K1OM/glibc-extra-nss-*.*.k1om.rpm micN:/tmp
   ```

   and `rpm` installed:

   ```bash
   [micN]# rpm -ivh /tmp/rpcbind-0.*.k1om.rpm
   [micN]# rpm -ivh /tmp/yp-tools-*.*.k1om.rpm
   [micN]# rpm -ivh /tmp/ypbind-mt-*.*.k1om.rpm
   [micN]# rpm -ivh /tmp/glibc-extra-nss-2.*.k1om.rpm
   ```
2. Start the rpcbind daemon.

   [micN]# /etc/init.d/rpcbind start

3. Add the NIS/YP server to /etc/yp.conf and start the ypbind daemon.

   [micN]# echo "domain <domain name> server <server IP address>"
   >>/etc/yp.conf
   [micN]# domainname <domain name>
   [micN]# /etc/init.d/ypbind start

4. Configure nss.

   [micN]# cat <<EOF >>/etc/nsswitch.conf
   passwd: nis files
   shadow: nis files
   group: nis files
   EOF

5. Configure ssdh.

   [micN]# echo "UsePAM yes" >>/etc/ssh/sshd_config

6. Configure PAM.

   [micN]# sed -ie"s/pam_unix.so/pam_unix.so nis/" /etc/pam.d/common-auth
   [micN]# sed -ie"s/pam_unix.so/pam_unix.so nis/" /etc/pam.d/common-account
   [micN]# sed -ie"/session/s/required/optional/" /etc/pam.d/sshd

7. Restart ssdh (the above changes will take effect).

   [micN]# /etc/init.d/sshd restart

6.2.1.3 Enabling NFS auto mount with the NIS/YP service

autofs can be installed on a coprocessor and configured to dynamically mount the NIS server. The following steps modify steps 1 and 7 in the previous Section 6.2.1.2:

1. Copy and install additional RPMs:

   [host]$ scp $MPSS38_K1OM/nfs-utils-client-*.klom.rpm mic0:/tmp
   [host]$ scp $MPSS38_K1OM/autofs-5.*.klom.rpm mic0:/tmp

   and rpm install them:

   [micN]# rpm -ivh /tmp/nfs-utils-client-*.klom.rpm
   [micN]# rpm -ivh /tmp/autofs-5.*.klom.rpm

7. After configuring PAM, and before restarting ssdh, configure autofs and re-start the autofs/automount daemon:

   [micN]# echo "/home /etc/auto.misc " >>/etc/auto.master
6.2.2 Enabling SSH host based authentication

1. Configure `sshd` to enable host based authentication.

   ```
   [host]# cat <<EOF >>/etc/ssh/sshd_config
   HostbasedAuthentication yes
   IgnoreRhosts no
   EOF
   ```

2. Register SSH client to a user.

   ```
   [host]# cat <<EOF ><home directory>/.shosts
   <server IP address>
   EOF
   [host]# chmod 600 <home directory>/.shosts
   [host]# chown <owner:group> <home directory>/.shosts
   ```

3. Create an entry for SSH client in user's `known_hosts`.

   ```
   [host]# ssh -X <user>:<HostBasedAuthClient>
   Are you sure you want to continue connecting (yes/no)? yes
   <user>@<server IP address>’s password:
   ^D
   exit
   ```

4. Restart SSH daemon.

   ```
   [mic]# /etc/init.d/sshd restart
   ```

5. Remove SSH key to ensure that user based authentication is not used.

   ```
   [host]$ cd <home directory>/.ssh
   [host]$ rm -f authorized_keys id_rsa*
   ```
7 Adding software to the coprocessor’s file system

Typical installations are not static, and often require the system administrator to add additional files or directories to the Intel® Xeon Phi™ coprocessor’s root file system. This section describes a range of techniques and considerations for performing such additions.

The mpss-3.8-k1om.tar file, which can be obtained from the Intel® Developer Zone website (Intel® DZ), is composed of over 1900 RPM packages built for installation into the coprocessor's file system. This section will describe options for installing these RPMs. If a component or application is not included in mpss-3.8-k1om.tar, refer to Section 8 to learn how to build software packages for the coprocessor.

Most software can be added to a file system while it is resident on the host or another node from which it is NFS exported, or while it is resident on the coprocessor. In all of these cases, the software to be added might be in the form of an RPM, a tarred installation package, or another form.

7.1 Adding individual files to a host resident file system image

7.1.1 Assisted configuration

The process of creating the file system image is driven by the Base, CommonDir, MicDir, and Overlay configuration parameters. These were previously described in Section 4.1.1.3. The overlay process can be used to add individual files as well as directory hierarchies to the coprocessor file system. Software added to directories indicated by these parameters is persistent across reboots.

For example, assume that you have cross-compiled an autotools-based software package (cross compiling is discussed in Section 8.1). The last step in that process is to make install the resulting components. One option is to make install into the CommonDir overlay directory. The CommonDir parameter syntax is:

\[
\text{CommonDir <source>}
\]

Assuming that the CommonDir parameter has the value /var/mpss/common, for example:

\[
\text{CommonDir /var/mpss/common}
\]

then the command:

\[
\text{[host]# make install DESTDIR=/var/mpss/common}
\]

installs the component into that overlay. On booting the software will be available on all coprocessors because the CommonDir overlay is common to all coprocessors.
Adding software to the coprocessor’s file system

The Overlay parameter:

Overlay (Filelist|Simple|File) <source> <target> (on|off)
Overlay RPM <source> (on|off)

can be used to add software to the coprocessor file system. The Overlay parameter(s) can be unique to each coprocessor.

As an example of using the Overlay Simple option, you could perform the sequence:

[host]$ mkdir <component>
[host]# make install DESTDIR=<component>

to install software into a directory that is specific to that component, and then use the Simple overlay type to add the component to the coprocessor file system:

[host]# micctrl --overlay=simple --source=<component>/ * \ 
--target=/ --state=on

In this way, a collection of components can be built, each in its own directory, which can be selectively added to the coprocessor file system:

[host]# micctrl --overlay=simple --source=<component1>/ * \ 
--target=/ --state=on
[host]# micctrl --overlay=simple --source=<component2>/ * \ 
--target=/ --state=on
 : 
[host]# micctrl --overlay=simple --source=<componentN>/ * \ 
--target=/ --state=on

Note: The filelist overlay type might be deprecated in a future Intel® MPSS release. Please use the simple and file overlay types instead.

7.1.2 Manual configuration

When performing a manual configuration, you can install individual files or tarred groups of files directly into a coprocessor’s file system hierarchy. Assuming the file system is maintained as a compressed CPIO archive (the form in which it is installed), you must first expand it. Here we expand the installed file system image to <some directory>:

[host]$ mkdir <some directory>; cd <some directory>
[host]# gunzip -c /usr/share/mpss/boot/ \ 
inframfs-knightscorner.cpio.gz | cpio -ivd

After adding software, and if the file system is to be pushed to the coprocessor’s memory it must first be re-archived and compressed:

[host]$ cd <some directory>
[host]# find . | cpio -o -H newc | gzip > <some_ramfs.cpio.gz>

Boot the coprocessor(s) specifying this new file system image as described in Section 4.2.5. If the coprocessor’s file system is to be NFS mounted, then there is no need to perform the last step.
### 7.1.3 Installing RPMs

Many of the RPMs in the *mpss-3.8-k1om.tar* file can be installed into the coprocessor’s file system while it is resident on the host. Some RPM installations require execution of a binary, such as a program to validate the installation. Since RPMs in the *mpss-3.8-k1om.tar* file are built for execution on the coprocessor, no such binary can be executed on an Intel® Xeon® processor-based host. Generally, library RPMs can be installed on the host, while application RPMs are more likely to require installation on the coprocessor.

As discussed in more detail in [Section 8](#), the RPM database in the default file system is built with rpm v5, which should thus be used to add software to that file system. The MPSS SDK includes an rpm v5 implementation that can be used for that purpose. Sourcing the file `/opt/mpss/3.8/environment-setup-k1om-mpss-linux` prepends your PATH with `/opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin` so that the `rpm` command resolves to the rpm v5 executable in the Intel® MPSS SDK. It’s recommended to use `su` to become root when doing this.

```
[host]$ su
[host]# source /opt/mpss/3.8/environment-setup-k1om-mpss-linux
```

**Note:** The resulting PATH will cause other binaries, such as python, to be found in `/opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin`. It is therefore recommended that `/opt/mpss/3.8/environment-setup-k1om-mpss-linux` is only sourced into the environment in which cross compilation is being performed.

Verify that you will execute the `rpm` from the Intel® MPSS sdk:

```
[host]# which rpm
/opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin/rpm
```

### 7.1.3.1 Assisted configuration

RPMs can be installed into a Base file system, for example into the default file system image that is installed at `/usr/share/mpss/boot/initramfs-knightscorner.cpio.gz`. The default file system includes the database of RPMs that are already installed.

Use the `micctrl --base` command to extract the default file system compressed CPIO image to some directory:

```
[host]# micctrl --base=DIR --new=<some directory>
```

**Note:** `micctrl` only extracts files to `<some directory>` if that directory does not already exist. If `<some directory>` already exists, `micctrl` will only change the Base configuration parameter.

You can now install k1om RPMs into the file system at `<some directory>`. For example:

```
[host]# rpm --root=<some directory> --dbpath=/var/lib/rpm -i $MPSS3B_K1OM/<some.rpm>
```

The `--dbpath` option tells `rpm` to use the database at `/var/lib/rpm` relative to `--root`. Thus it will use the RPM data base in the coprocessor’s file system.
Adding software to the coprocessor’s file system

You can leave the file system in `<some directory>`. It will be used when constructing either an NFS mounted file system or ram file system according to the `RootDevice` parameter. For example:

```bash
[host]# micctrl --rootdev=NFS -c
```

builds the NFS exported file system using the `Base` at `<some directory>`. Alternatively, if `RootDevice` is set to RamFS:

```bash
[host]# micctrl --rootdev=RAMFS
```

then the CPIO image will be built at boot time from the file system at `<some directory>`.

### 7.1.3.2 Manual configuration

If doing Manual Configuration, expand a CPIO compressed image such as the default file system `/usr/share/mpss/boot/initramfs-knightscorner.cpio.gz`:

```bash
[host]$ mkdir <some directory>; cd <some directory>
[host]# gunzip -c <some ramfs> | cpio -ivd
```

Install k1om RPMs as needed:

```bash
[host]$ rpm --root=<some directory> --dbpath=/var/lib/rpm \ -i $MPSS38_K1OM/<some.rpm>
```

If not NFS exporting the file system, re-archive the image:

```bash
[host]$ cd <some directory>
[host]# find . | cpio -o -H newc | gzip > <some other ramfs>
```

Boot as described in Section 4.2.5.

### 7.2 Adding software to the coprocessor’s file system

Installing software to the coprocessor’s file system while mounted on the coprocessor is much like adding software to any Linux* file system. Individual files can be directly copied to the target directory on the coprocessor using `scp`. Tar files can be copied to the coprocessor using `scp` and untarred into the appropriate directory.

The rest of this section discusses different ways to install RPMs and how to preserve the modified file system.

#### 7.2.1 Installing RPMs

To install RPMs into a coprocessor mounted file system, you can use one of the following procedures.

**Note:** These instructions assume that `mpss-3.8-k1om.tar` has been untarred to some `$MPSS38_K1OM` directory. The `mpss-3.8-k1om.tar` is available for download at the
7.2.1.1 Using the overlay RPM configuration parameter and the micctrl --overlay utility

The Overlay RPM configuration parameter has the form:

Overlay RPM <source> (on|off)

If <source> is an RPM file, then it is copied to a special /RPMs-to-install directory in the file system image that is pushed to one or more coprocessors, depending on whether the parameter is added to micN.conf or the default.conf configuration file. If <source> is a directory, then all the RPM files in that directory are copied to /RPMS-to-install. Multiple Overlay parameters are allowed. These parameters can be edited directly or the micctrl --overlay command can be used to add, modify or remove such parameters; see Appendix A.4.2 and Appendix B.4.4.4 for additional details.

The RPMs in /RPMS-to-install are installed during the early phase of booting a coprocessor. Some RPMs cannot be successfully installed during early boot phase, due to dependencies that cannot be satisfied during that phase. If RPMs are not installed successfully, log files (refer to Appendix I.1) may provide helpful information.

7.2.1.2 Manually installing RPMs using scp

An alternative is to copy RPMs to the coprocessor’s file system and rpm install them.

SCP the RPMs to the coprocessor:

```
[host]$ scp <rpm_packages> micN:<some directory>
```

SSH to the coprocessor as root:

```
[host]$ ssh micN
```

Install the RPMs using the rpm utility:

```
[micN]$ cd <some directory>
[micN]$ rpm -ihv <rpm_packages>
```

For example, to install man, copy the man RPM to a coprocessor:

```
[host]$ scp $MPSS38_K1OM/man-1.6f-r2.k1om.rpm micN:/tmp
man-1.6f-r2.k1om.rpm 100% 130KB 129.7KB/s 00:00
```

Attempt to install the RPM. Here we assume that /tmp only holds RPM files that we have copied for this example:

```
[micN]$ cd /tmp
[micN]$ rpm -ihv *.rpm
error: Failed dependencies:
  less is needed by man-1.6f-r2.k1om
  groff is needed by man-1.6f-r2.k1om
```
By iteratively copying RPMs and attempting to install them, less, groff, perl and libperl5 RPMs are copied to the coprocessor, where installation can now complete successfully:

```
[micN]# rpm -ihv *.rpm
Preparing... #............................................................
[100%]
1:libperl55 #............................................................
[ 20%]
2:man #............................................................
[ 40%]
3:less #............................................................
[ 60%]
update-alternatives: Linking /usr/bin/less to less.less
4:perl #............................................................
[ 80%]
5:groff #............................................................
[100%]
```

### 7.2.1.3 Installing RPMs with zypper using an http repo

One obvious disadvantage of the previous method is that, where there are dependencies, the user must install RPMs in the correct order. This can be solved by creating a repo on the host that zypper can access from the coprocessor. Zypper is preinstalled in the coproessor’s default file system.

The steps in this section are for creating a repository of RPMs and using the Python SimpleHTTPServer for serving them; we assume that these tools have been previously installed on the host. Though other repository creation tools and HTTP servers are available, we only provide instructions for using createrepo and Python SimpleHTTPServer. The host firewall or iptables may need to be configured to allow zypper to access the repository on the host.

Change to the folder where the k1om RPMs were extracted:

```
[host]$ cd $MPSS38_K1OM
```

Use the createrepo tool to create a new repo:

```
[host]$ createrepo .
```

Start an http server as follows:

```
[host]$ python -m SimpleHTTPServer ${PORT_NUMBER}
```

From another terminal, add the repo on a coprocessor:

```
[host]$ ssh root@micN -R $(SOME_PORT):host:${PORT_NUMBER}
[micN]$ zypper addrepo http://host:${PORT_NUMBER} mpss
```

If no port is specified, `python -m SimpleHTTPServer` defaults to port 8000. In that case, the following is sufficient:

```
[host]$ ssh root@micN
```
Adding software to the coprocessor's file system

```
[micN]# zypper addrepo http://host:8000 mpss
```

Now install RPMs as needed:

```
[micN]# zypper install <rpm file>
```

For example, to install `man`:

```
[micN]# zypper install man
File 'repomd.xml' from repository 'mpss' is unsigned, continue? [yes/no] (no): yes
Building repository 'mpss' cache [done]
Loading repository data...
Reading installed packages...
Resolving package dependencies...
The following NEW packages are going to be installed:
  groff less libperl5 man perl
5 new packages to install.
Overall download size: 2.8 MiB. After the operation, additional 8.4 MiB will be used.
Continue? [y/n/?] (y): y
Retrieving package libperl5-5.14.2-r7.k1om (1/5), 709.0 KiB (1.5 MiB unpacked)
Retrieving: libperl5-5.14.2-r7.k1om.rpm [done]
Retrieving package less-444-r2.k1om (2/5), 78.0 KiB (163.0 KiB unpacked)
Retrieving: less-444-r2.k1om.rpm [done]
Retrieving package perl-5.14.2-r7.k1om (3/5), 16.0 KiB (36.0 KiB unpacked)
Retrieving: perl-5.14.2-r7.k1om.rpm [done]
Retrieving package groff-1.20.1-r1.k1om (4/5), 1.9 MiB (6.4 MiB unpacked)
Retrieving: groff-1.20.1-r1.k1om.rpm [done]
Retrieving package man-1.6f-r2.k1om (5/5), 130.0 KiB (266.0 KiB unpacked)
Retrieving: man-1.6f-r2.k1om.rpm [done]
Installing: libperl5-5.14.2-r7 [done]
Installing: less-444-r2 [done]
Additional rpm output:
  update-alternatives: Linking //usr/bin/less to less.less
  Installing: perl-5.14.2-r7 [done]
  Installing: groff-1.20.1-r1 [done]
  Installing: man-1.6f-r2 [done]
```

We see that `zypper` takes care of all the dependencies if they can be satisfied by the RPM files in the repo.

The directory containing such a repository can also be NFS mounted. `Zypper` can then access it as in a local directory.

### 7.2.2 Preserving the modified file system

An important consideration in adding software to a file system in coprocessor memory is persistence. Assuming the file system is exported from a permanent storage device,
modifications to an NFS mounted file system are persistent. Conversely, when the file system is in coprocessor memory (RAMFS), any modifications to the file system are lost when the coprocessor is shut down unless steps are taken to capture the file system image to permanent storage.

The following command, executed from the host, captures the current file system of a specified coprocessor to host file /usr/share/mpss/boot/custom.cpio.gz:

```
[host]# ssh root@micN 'cd / ; find . /dev ! -xdev ! -path "/.etc/modprobe.d*" ! -path "/.var/volatile/run*" | cpio -o -H \
newc | gzip -9' > /usr/share/mpss/boot/custom.cpio.gz
```

**Note:** /dev is specified as a path because the -xdev option would otherwise prevent capturing that path.

To use the captured file system image, change the RootDevice parameter to StaticRamFS, and target the captured file. For example:

```
[host]# micctrl --rootdev=StaticRamFS \
    --target=/usr/share/mpss/boot/custom.cpio.gz
```

and restart the coprocessor:

```
[host]# micctrl -Rw
```

If performing manual configuration, the captured image is specified in the boot string. For example:

```
[host]# micctrl -rw
[host]# echo \n"boot:linux:/usr/share/mpss/boot/bzImage-knightscorner: \
/usr/share/mpss/boot/custom.cpio.gz" > /sys/class/mic/micN/state
[host]# micctrl -b
```

See Section 4.2.5 for details on manual configuration control of the coprocessor.
# 8 Compilation for the coprocessor

This section takes you through using some of the techniques and tools that you will need to know in order to compile software for native execution on the Intel® Xeon Phi™ coprocessors. This section begins with a description of how to cross compile software using the Intel® MPSS SDK. We then discuss native compilation on the coprocessor.

*Note:* This document does not cover Intel® Composer XE support of offload programming. There are numerous other documents and books that cover this topic. For more information, see Programming and Compiling for Intel® Many Integrated Core Architecture.

## 8.1 Cross compiling software with the Intel® MPSS SDK

Intel® MPSS includes an SDK that supports cross compilation of software for execution on the coprocessor. In this section we will discuss the components of the SDK, and general recommendations for cross compiling software components to be added to the coprocessor file system. We will illustrate this process by building the `zsh` shell.

### 8.1.1 SDK overview

The cross-compilation SDK is installed at `/opt/mpss/3.8/sysroots/x86-64-mpsssdk-linux`. It includes `gcc` cross compiler as well as standard utilities such as `ar`, `as`, `ld`, `nm`, `objcopy`, `objdump`, `ranlib`, `rpm` (v5) and `strip`. Generally speaking, the SDK only includes tools that must be aware of the format of the coprocessor binary executables. For example, `make` has no dependence on binary executable formats, and thus does not need to be in the SDK; the version of `make` that is installed on the build machine can be used.

The `/opt/mpss/3.8/sysroots/k1om-mpss-linux` subtree contains header files and libraries that are built for the coprocessor, and that are expected to be needed during the build process. Additional dependencies can be added as illustrated later in an example.

The RPM databases in both `sysroots`:

- `/opt/mpss/3.8/sysroots/k1om-mpss-linux/var/lib/rpm`
- `/opt/mpss/3.8/sysroot/x86-64-mpsssdk-linux/var/lib/rpm`

and the default `initramfs` are in rpm v5 format. This format differs from the format generated by rpm v4. Therefore rpm v5 (rpm5) must be used to install packages into `/opt/mpss/3.8/sysroots/k1om-mpss-linux` and into the coprocessor file system. The `/opt/mpss/3.8/environment-setup-k1om-mpss-linux` script sets PATH so that rpm v5, as well as the other sdk utilities mentioned above, can be found.

### 8.1.2 Cross compilation of gnu build system based packages

The GNU Build System, also known as the Autotools, is part of the GNU toolchain that is used for making source code packages portable to a wide range of Unix*-like systems. A
A vast number of source code packages are based on the GNU Build System. In fact, virtually every RPM package in mpss-3.8-k1om.tar was built from an open source GNU Build System source code package.

On many platforms, native compilation - building a GNU Build System package for execution on the same platform - only requires unpacking the package, and changing to the newly created directory to run the configure script. The script probes the system for various features to create a makefile needed to build the package on the local system. Afterwards make is executed to create libraries, executables and other files that comprise the package, followed by make install to copy the resulting files to their proper location on the system. In the case of native compilation, the configure script can determine the compiler and other build tools to use by probing the system.

When cross compiling a GNU Build System package, however, configure must be told explicitly about the build platform: where compilation is performed, and the host platform: where the executable will be run. This is done via the configure options:

- The system on which the package is built.
  
  --build=build

- The system where built programs and libraries will run.
  
  --host=host

- When building compiler tools, the system for which the tools will create output:
  
  --target=target

Specifying the --host option tells configure that this is a cross compilation build. The script, in turn, searches for the cross-compiling suite for the named host platform. In the case of cross-compiling for the coprocessor on an x86_64 Linux* platform, the following configure options are required:

  --build=x86_64-linux
  --host=k1om-mpss-linux
  --target=k1om-mpss-linux

Cross-compilation tools commonly have their target architecture as a prefix of their name, thus configure will search for k1om-mpss-linux-gcc, etc.

The /opt/mpss/3.8/environment-setup-k1om-mpss-linux script, mentioned earlier, sets up for GNU Build System based builds, defining environment variables as needed by configure and make, and by rpm installation into the SDK. In particular, it defines the configure options described above and prepends to PATH such that configure will find the cross tools.

**Note:** The resulting PATH will cause certain binaries, such as python, to be found in /opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin. It is therefore recommended that only cross-compilation be performed in an environment in which /opt/mpss/3.8-environment-setup-k1om-mpss-linux has been sourced. This will avoid executing a binary in /opt/mpss/3.8/sysroots/x86_64-mpsssdk-linux/usr/bin when it was intended to execute the version installed on the host, for example in /usr/bin.
8.1.3 Example case: zsh

`mpss-3.8-k1om.tar` includes prebuilt RPM packages for many components. These packages can be directly installed into the coprocessor file system as described in Section 7. It does not, however, include a package for the `zsh` shell. To illustrate the cross-compilation process, we will step through building `zsh`, and installing it into the coprocessor file system.

**Note:** We assume that the RPMs in `mpss-3.8-k1om.tar` were extracted to `$MPSS38_K1OM`.

1. Download and untar the `zsh` source distribution.

   ```bash
   [host]$ tar xvf zsh-5.0.5.tar.bz2
   [host]$ cd zsh-5.0.5
   [host]$ export ZSH=`pwd`
   ```

2. Setup the environment, and try to generate a makefile.

   Source the `environment-setup-k1om-mpss-linux` script and invoke `gnu-configize`:

   ```bash
   [host]# source /opt/mpss/3.8/environment-setup-k1om-mpss-linux
   [host]# gnu-configize
   ```

   `gnu-configize` is an Autotools generated binary whose purpose is to update `config.sub`. For most software packages it is sufficient to teach it to understand '--host=k1om-mpss-linux'. If `gnu-configize` does not work, the most likely reason is that the software package was not generated by autotools—perhaps the `configure` script was hand-written; in this case it is necessary to read the code to determine how to properly cross compile it. Cross compiling software that does not use autotools is beyond the scope of this document.

   Now invoke the `zsh` `configure` script. In this example, `configure` fails due to an unresolved dependency on `ncurses` and `ncurses_devel`. The output from `configure` is abbreviated:

   ```bash
   [host]# ./configure $CONFIGURE_FLAGS --prefix=/usr --libdir=/usr/lib64
   configure: WARNING: unrecognized options: --with-libtool=sysroot
   configure: loading site script /opt/mpss/3.8/site-config-k1om-mpss-linux
   : checking for library containing tigetflag... no
   checking for library containing tgetent... no
   configure: error: in '/home/mic/Downloads/zsh-5.0.5':
   configure: error: "No terminal handling library was found on your system.
   This is probably a library called 'curses' or 'ncurses'. You may
   need to install a package called 'curses-devel' or 'ncurses-devel'
   on your system."
   See 'config.log' for more details
   ```

3. Resolve dependency issues.

   The next step is to satisfy dependencies; this is an iterative process. Prebuilt packages for both `ncurses` and `ncurses-devel` are included in `mpss-3.8-k1om.tar`.
Install ncurses and ncurses-dev:

[host]\# rpm --root $OECORE_TARGET_SYSROOT --dbpath /var/lib/rpm -i \\ $MPSS38_K1OM/ncurses-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/ncurses-dev-5.9-r8.1.klom.rpm \\
error: Failed dependencies:
  libform5 is needed by ncurses-dev-5.9-r8.1.klom
  libtic5 is needed by ncurses-dev-5.9-r8.1.klom
  libpanel5 is needed by ncurses-dev-5.9-r8.1.klom
  libmenu5 is needed by ncurses-dev-5.9-r8.1.klom
    /usr/lib64/libform.so.5 is needed by ncurses-dev-5.9-r8.1.klom
    /usr/lib64/libmenu.so.5 is needed by ncurses-dev-5.9-r8.1.klom
    /usr/lib64/libpanel.so.5 is needed by ncurses-dev-5.9-r8.1.klom
    /usr/lib64/libtic.so.5 is needed by ncurses-dev-5.9-r8.1.klom

We now see that libform5, libtic5, libpanel5, and libmenu5 are also needed. Adding them to the command reveals that libncurses is also needed:

[host]\# rpm --root $OECORE_TARGET_SYSROOT --dbpath /var/lib/rpm -i \\ $MPSS38_K1OM/ncurses-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/ncurses-dev-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libform5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libtic5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libpanel5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libmenu5-5.9-r8.1.klom.rpm \\
error: Failed dependencies:
  libncurses5 >= 5.9 is needed by libform5-5.9-r8.1.klom
  libncurses.so.5()(64bit) is needed by libform5-5.9-r8.1.klom
  libncurses5 >= 5.9 is needed by libpanel5-5.9-r8.1.klom
  libncurses.so.5()(64bit) is needed by libpanel5-5.9-r8.1.klom
  libncurses5 >= 5.9 is needed by libmenu5-5.9-r8.1.klom
  libncurses.so.5()(64bit) is needed by libmenu5-5.9-r8.1.klom

When libncurses is added to the command, it reveals that pkgconfig is needed:

[host]\# rpm --root $OECORE_TARGET_SYSROOT --dbpath /var/lib/rpm -i \\ $MPSS38_K1OM/ncurses-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/ncurses-dev-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libform5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libtic5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libpanel5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libmenu5-5.9-r8.1.klom.rpm \\ $MPSS38_K1OM/libncurses5-5.9-r8.1.klom.rpm \\
error: Failed dependencies:
  pkgconfig is needed by ncurses-dev-5.9-r8.1.klom
When `pkgconfig` is added to the command, we see that `libpopt0` is also needed:

```bash
[host]# rpm --root $OECORE_TARGET_SYSROOT --dbpath /var/lib/rpm -i \
$MPSS38_K1OM/ncurses-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/ncurses-dev-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libform5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libtic5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libpanel5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libmenu5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libncurses5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/pkgconfig-0.25-r3.k1om.rpm
```

error: Failed dependencies:

- `libpopt0 >= 1.16` is needed by `pkgconfig-0.25-r3.k1om`
- `libpopt.so.0(64bit)` is needed by `pkgconfig-0.25-r3.k1om`
- `libpopt.so.0(LIBPOPT_0)(64bit)` is needed by `pkgconfig-0.25-r3.k1om`

So we add `libpopt0` to the command, and `rpm` installation completes:

```bash
[host]# rpm --root $OECORE_TARGET_SYSROOT --dbpath /var/lib/rpm -i \
$MPSS38_K1OM/ncurses-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/ncurses-dev-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libform5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libtic5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libpanel5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libmenu5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/libncurses5-5.9-r8.1.k1om.rpm \
$MPSS38_K1OM/pkgconfig-0.25-r3.k1om.rpm \
$MPSS38_K1OM/libpopt0-1.16-r0.k1om.rpm
```

Now, try to configure the build again (abridged output shown):

```bash
[host]$ cd $ZSH
[host]$ ./configure $CONFIGURE_FLAGS --prefix=/usr \
--libdir=/usr/lib64
```

configure: WARNING: unrecognized options: --with-libtool
configure: loading site script /opt/mpss/3.8/site-config-kom-mpss-linux
configuring for zsh 5.0.5

```
zsh configuration
---------------
zsh version : 5.0.5
host operating system : klm-mpss-linux-gnu
source code location : .
compiler : klm-mpss-linux-gcc
preprocessor flags : -m64 --
sysroot=/opt/mpss/3.8/sysroots/klm-mpss-linux
executable compiler flags : -m64 --
sysroot=/opt/mpss/3.8/sysroots/klm-mpss-linux
```
Compilation for the coprocessor

Executable linker flags: 
- sysroot=/opt/mpss/3.8/sysroots/kлом-mpss-linux -rdynamic
- library flags: -ldl -ltinfo -lrt -lm -lc
- installation basename: zsh
- binary install path: /usr/bin
- man page install path: ${prefix}/share/man
- info install path: ${prefix}/share/info
- functions install path: ${prefix}/share/zsh/5.0.5/functions

See config.modules for installed modules and functions.

Configuration was successful, and a Makefile has been generated.

4. Build the Binaries with the Generated Makefile

You can now build zsh:

[host]$ make
make[1]: Entering directory `/home/mic/Downloads/zsh-5.0.5/Src'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory `/home/mic/Downloads/zsh-5.0.5/Doc'

5. Install the Component

You are now ready to install zsh. Where and how you install it depends to some extent on whether you are doing assisted configuration or manual configuration (see Section 4), and other considerations. If doing assisted configuration, one approach is to install the files into the CommonDir overlay directory, that is, the directory identified by the CommonDir configuration parameter. On booting a coprocessor, the CommonDir directory overlays the Base file system of each coprocessor (see Section 4.1.1). Here we assume the default location for CommonDir:

[host]$ make install DESTDIR=/var/mpss/common

If doing manual configuration, you might install directly into a file system image. To do this, you must first expand the compressed cpio archive. Here we expand some ramfs image to $HOME/initramfs:

[host]$ mkdir $HOME/initramfs; cd $HOME/initramfs
[host]$ gunzip -c <current ramfs_location> | cpio -ivd

where <current ramfs_location> is the path to some compressed CPIO file system archive.

Then install zsh into the file system image:

[host]$ make install DESTDIR=$HOME/initramfs

Finally re-archive the file system image:

[host]$ find . | cpio -o -H newc | gzip > <new ramfs_location>
where `<new_ramfs_location>` is the name of the file system archive to be created.

Of course, you can also install into an empty directory and then tar the resulting hierarchy for later use.

### 8.1.4 Cross compiling with ICC

For many tools and components, such as `zsh`, execution performance is not critical, and cross compiling with `gcc` is recommended. However, when building performance critical applications, cross compiling with `icc` is likely to result in better performance. Generally speaking, if an application can be compiled for x86_64 using `icc`, then it can be cross compiled for the coprocessor using `icc`. Since such performance sensitive applications are almost always ported to x86_64 and `icc`, it follows that most such applications can be cross compiled for execution on the coprocessor.

The process for cross compiling with `icc` is as described above except that some variables need to be modified when calling `configure`:

- `CC` needs to be set to `icc`
- `CFLAGS` needs to be extended by `-mmic`
- `CXXFLAGS` needs to be extended by `-mmic`

A complete example might look like:

```bash
[host] $ ./configure $CONFIGURE_FLAGS --prefix=/usr \ 
   --libdir=/usr/lib64 LDFLAGS="'LD=k1om-mpss-linux-ld \ 
   CPPFLAGS="' CC=icc CFLAGS='-mmic -I/opt/include' CXX=icpc \ 
   CXXFLAGS='-mmic'
```

There is no special cross compiling version of `icc` that generates code for the coprocessor. Instead, the `-mmic` option to `icc` instructs it to cross compile for the coprocessor. Therefore `icc` is not included in the Intel® MPSS SDK, but rather must be installed on the host as part of Intel® Composer XE installation. The `k1om-mpss-linux-ld` cross linker is still needed.

### 8.2 Native compilation

Native compilation on the coprocessor can often be easier than cross compilation. As we did for cross compilation, we focus here on building software packages created using the autotools.

As mentioned previously, the coprocessor does not have a hard disk based file system, so all tools, source code and temporary files need to fit into its memory. Large projects might require you to use alternatives, such as an NFS mounted file system.

Because there is no native version of the Intel® ICC compiler, native compilation uses `gcc` and is thus generally limited to components that are not particularly performance sensitive; performance oriented applications should be cross compiled using the Intel® ICC compiler.
In order to perform native compilation, gcc, the GNU binutils, and other common development tools must be installed into the coprocessor’s file system. These components are not already installed in the initramfs to save space. This is performed by installing a single RPM, `task-mpss-toolchain`.

You can then perform the same autotools build process previously described, installing additional dependent RPMs as needed. When you have successfully built the component, you will have to decide what you will do with the build results. For example, `make installing` the result into a local ramfs file system is not persistent unless the resulting file system image is captured for subsequent reuse. Alternatively, `make installing` the result into an NFS mounted file system captures the component for subsequent invocations.

To illustrate native compilation, it will be shown how to build and install `emacs`. It is assumed that coprocessors, on which the built will be performed, are in the online state.

### 8.2.1 Creating and adding a repo

The first step is to create a repository of the RPMs in the `mpss-3.8-k1om.tar` and start an http server to serve the repo data:

```bash
[host]$ tar -xf mpss-3.8-k1om.tar
[host]$ cd $MPSS38_K1OM
[host]$createrepo .
[host]$python -m SimpleHTTPServer
```

Next, ssh to the selected coprocessor and add the repo:

```bash
[host]$ ssh root@micN
[mic]$ zypper ar http://host:8000
```

### 8.2.2 Installing the development tool chain

The default coprocessor file system does not include software development tools, so these must now be installed:

```
[micN]$ zypper install task-mpss-toolchain
```

Installing the `task-mpss-toolchain-3.8-0.1.rc4.all.rpm` causes the following components to be installed as dependencies:

- `cpp-symlinks`, `flex`, `byacc`, `cmake`, `makedepend`, `gperf`, `g++-symlinks`, `gcov-symlinks`, `gnu-config`, `pkgconfig`, `patch`, `automake`, `m4`, `bison`, `gccmakedep`, `gcc-symlinks`, `autoconf`, `libtool`, `elfutils`, `binutils-symlinks`, and `make`

Because building `task-mpss-toolchain-3.8-0.1.rc4.all.rpm` takes considerable time and adds significantly to the size of the file system, consider capturing the file system at this point for later reuse in building other components. See Section 7.1 and 7.2 for help.

### 8.2.3 Configuring the build directory

Next, from the host, copy the software package that is to be built to the coprocessor file system. We will build `emacs` to illustrate this process:
[host]$ scp emacs-24.3.tar.gz mic0:/tmp

Now, on the coprocessor, try to configure the build directory:

[host]$ ssh micN
[micN]$ cd /tmp
[micN]$ tar xvf emacs-24.3.tar.gz
[micN]$ cd emacs-24.3
[micN]$ gnu-configize
[micN]$ ./configure --prefix=/usr --libdir=/usr/lib64
checking for a BSD-compatible install... build-aux/install-sh -c checking whether build environment is sane... yes :
configure: error: The required function `tputs' was not found in any library.
The following libraries were tried (in order):
  libtinfo, libncurses, libterminfo, libtermcap, libcurses
Please try installing whichever of these libraries is most appropriate
for your system, together with its header files.
For example, a libncurses-dev(el) or similar package.

Just as in cross-compiling zsh, we need to install ncurses. Assuming that the repo is still attached, this can be done easily with zypper (as root):

[micN]# zypper install ncurses-dev

We can now successfully configure the build directory:

[micN]$ ./configure --prefix=/usr --libdir=/usr/lib64

*Note:* In the case that configure has failed, it is sometimes helpful to execute:

[micN]$ make distclean

before executing configure again.

### 8.2.4 Making and installing the package

Make the software package:

[mic]$ make

You can now just install the package into the current file system:

[mic]$ make install

Alternatively, you can capture the results for later installation into some other file system image. For example, the following installs the package to some specified subdirectory, then creates a tarfile, `emacs.tar`, of that subdirectory:

[mic]$ make install DESTDIR=`pwd`/tarhere
[mic]$ tar -cf emacs.tar -C tarhere .
Copy the tar file to the host to save it:

[host]$ scp mic0:/tmp/emacs.tar .

The tar file can later be expanded onto an NFS mounted file system on the host, for example:

[host]# tar -xf emacs.tar -C /var/mpss/mic0.export

or copied to a coprocessor and expanded, for example:

[host]$ scp emacs.tar micN:/tmp
[host]$ ssh micN
[mic]$ cd /tmp
[mic]$ tar -xf emacs.tar -C /

§
9 Intel® MPSS component configuration and tuning

9.1 The coprocessor OS configuration and tuning

9.1.1 Clock source for the coprocessor

By default, the Time Stamp Counter (TSC) is the clock source on the Intel® Xeon Phi™ coprocessor. The power management software for the coprocessor will keep the TSC clock source calibrated even when deep sleep states are enabled. Calibration of the TSC avoids clock drift.

Each coprocessor core also has an Elapsed Time Counter (MICETC). However, when MICETC is the clock source, the gettimeofday() access time is on the order of 100x slower than when TSC is the clock source.

The available clock sources can be queried from sysfs on a coprocessor:

```
[micN]$ cat /sys/devices/system/clocksource/clocksource0/available_clocksource
tsc micetc
```

and the current clock source can be queried from sysfs, for example:

```
[micN]$ cat /sys/devices/system/clocksource/clocksource0/current_clocksource
tsc
```

The clock source can be changed by writing to sysfs, for example:

```
[micN]# echo micetc > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

9.1.2 Process oversubscription

Only configure concurrent processing when there is a real need for this feature. Otherwise, any workload running with the concurrent active processes on the device will likely result in performance degradation.

To run more concurrent processes, set the limit of file descriptors to 10 for each offload process. Depending on the memory usage of each process, a large number of concurrent offload processes may exhaust the device’s memory.

To run 200 concurrent processes, users will need to modify several parameters. Changes to the configuration will not persist when modifying the files directly on the coprocessor; a reboot will reset these settings. To permanently change the configuration, refer to the documentation on micctrl and file overlays.

1. On the host, log in to the coprocessor as superuser.
[host]# ssh micN

2. Locate and terminate the Intel® COI active process.

[micN]# ps axf | grep coi
5147 ? S 0:00 /usr/bin/coi_daemon --coiuser=micuser
[micN]# killall coi_daemon

3. Set the concurrent process to 200.

[micN]# ulimit -n 200
[micN]# /usr/bin/coi_daemon --coiuser=micuser --max-connections=200 &
[micN]# exit

For the complete list of \textit{coi}\_\textit{daemon} parameters, refer to the \textit{coi}\_\textit{daemon} help option:

[micN]$ /usr/bin/coi_daemon -h

\section*{9.1.3 Verbose logging}

Verbose output of coprocessor kernel boot messages can be disabled or enabled.

Assisted Configuration of verbose logging is controlled by the \textit{VerboseLogging} configuration parameter in /etc/mpss/default.conf or /etc/mpss/micN.conf configuration files:

VerbreoLogging <Disabled|Enabled>

The default is:

\begin{verbatim}
VerboseLogging Disabled
\end{verbatim}

For Manual Configuration, the \textit{quiet} kernel command line parameter disables verbose logging.

\section*{9.1.4 Cgroup memory control}

The cgroups memory controller can be disabled or enabled. The cgroups memory controller, when enabled, can limit the amount of memory available to an application or group of applications.

Assisted Configuration of cgroups is controlled by the \textit{Cgroup} parameter in /etc/mpss/default.conf or /etc/mpss/micN.conf configuration files:

\begin{verbatim}
Cgroup [memory=(disabled|enabled)]
\end{verbatim}

The default is:

\begin{verbatim}
Cgroup memory=disabled
\end{verbatim}

For Manual Configuration, the \textit{cgroup\_disable=memory} kernel command line parameter disables cgroups memory control. The absence of this parameter enables control.
9.1.5 Power management control

Power management control can be disabled or enabled.

Assisted Configuration of power management is accomplished by the presence of the PowerManagement parameter in /etc/mpss/default.conf or /etc/mpss/micN.conf configuration files:

```
PowerManagement cpufreq_(on|off);corec6_(on|off);
 pc3_(on|off);pc6_(on|off)
```

The default parameters vary depending on coprocessor stepping.

For Manual Configuration, the

```
 micpm=cpufreq_(on|off);corec6_(on|off);pc3_(on|off);pc6_(on|off)
```

kernel command line parameter controls power management.

**Note:** It is recommended to keep the default power management settings unless directed by an Intel® representative to change them.

9.1.6 VFS optimizations

As described in Section 0, a VFS technology preview is intended to improve the performance of system calls for reading and writing files on tmpfs and ramfs mount points. The following kernel command line options provide additional control to enable or disable the read and write optimizations:

- **vfs_read_optimization** - on/off. If not specified, it is off by default. When on, it enables read side optimizations for files in the above file systems.
- **vfs_write_optimization** - on/off. If not specified, it is off by default. When on, it enables write side optimizations for files in the above file systems.

As an example, to enable read optimizations, add `vfs_read_optimization` to the `ExtraCommandLine` as follows:

1. Edit `/etc/mpss/default.conf`
2. Append "`vfs_read_optimization=on`" to the `ExtraCommandLine` parameter.
3. Restart the `mpss` service

For more information on the `ExtraCommandLine` parameter, see Section A.3.1

9.2 Host driver configuration

9.2.1 Lost node watchdog

The host driver includes a watchdog intended to detect and report to the host when another coprocessor (node) in the SCIF network is not responding.
The watchdog is controlled by the "watchdog" parameter in the host's
/etc/modprobe.d/mic.conf module parameter control file.

If the host driver is loaded it must be reloaded. Follow the procedure:

```
[host]# service mpss unload
[host]# service mpss start
```

The watchdog is enabled by default.

### 9.2.2 Watchdog auto-reboot

On detecting a **lost** node, the host driver will either reset the node back to the **ready**
state, or reboot the node to the **online** state.

- Watchdog auto-reboot is controlled by the `watchdog_auto_reboot` parameter in the
  host's `/etc/modprobe.d/mic.conf` module parameter control file.

To change this parameter, the Intel® MPSS host driver must be reloaded if it is
currently running. Follow the procedure:

```
[host]# service mpss unload
[host]# service mpss start
```

Watchdog auto-reboot reboots the node to the online state by default.

### 9.2.3 Crash dump capture

The host driver can capture a coprocessor OS kernel crash dump to a file on the host.

- Crash dump capture is controlled by the `crash_dump` parameter in the host's
  `/etc/modprobe.d/mic.conf` module parameter control file.

To change this parameter, the Intel® MPSS host driver must be reloaded if it is
currently running. Follow the procedure:

```
[host]# service mpss unload
[host]# service mpss start
```

Crash dump capture is enabled by default.

### 9.3 SCIF configuration

#### 9.3.1 Peer to peer (p2p) support

SCIF supports the direct transfer of data from one coprocessor directly into the physical
memory of another coprocessor on the same host. This capability is referred to as Peer
to Peer or P2P.

- P2P is controlled by the `p2p` parameter in the host's `/etc/modprobe.d/mic.conf`
  configuration file.

To change this parameter, the Intel® MPSS host driver must be reloaded if it is
currently running. Follow this procedure:
9.3.2 Peer to peer proxy control

Under certain circumstances, SCIF implements peer-to-peer DMA reads (reading data from some remote peer coprocessor to the local coprocessor) into a peer-to-peer DMA write from the remote coprocessor to the local coprocessor. This is done to improve performance.

- P2P proxy is controlled by the \texttt{p2p\_proxy} parameter in the host's \\
  \texttt{/etc/modprobe.d/mic.conf} configuration file.
- To change this parameter, the Intel\textsuperscript{®} MPSS host driver must be reloaded if it is currently running. Follow this procedure:
  
  ```
  [host]\# service mpss unload
  [host]\# service mpss start
  ```

P2P proxy is enabled by default.

9.3.3 Ulimit checks for max locked memory in scif

SCIF can enforce \texttt{ulimit} checks of the memory that \texttt{scif\_register()} locks. Pages locked using \texttt{scif\_register()} are counted towards the \texttt{ulimit}.

- \texttt{Ulimit} checks are controlled by the \texttt{ulimit} parameter in the host's \\
  \texttt{/etc/modprobe.d/mic.conf} configuration file.
- To change this parameter, the Intel\textsuperscript{®} MPSS host driver must be reloaded if it is currently running. Follow this procedure:
  
  ```
  [host]\# service mpss unload
  [host]\# service mpss start
  ```

\texttt{Ulimit} checking is disabled by default.

\textbf{Note:} In kernel versions later than 3.1.0, the kernel has two different limits for locked pages: one limit for pages locked using standard system calls and another limit for pages locked by kernel modules on behalf of user processes.

9.3.4 Registration caching

\textbf{Note:} The mechanism for specifying the pinned pages limit may change in a future release.

Registration caching is a SCIF feature intended to improve the performance of \texttt{scif\_vreadfrom()}/\texttt{scif\_vwriteto()}. When registration caching is enabled, SCIF caches virtual to physical address translations of the virtual addresses passed to \texttt{scif\_vreadfrom()}/\texttt{scif\_vwriteto()}, thus eliminating the overhead of pinning pages when the same virtual range is specified in future calls.
Registration caching is controlled by the \textit{\texttt{reg\_cache}} parameter in the host’s \texttt{/etc/modprobe.d/mic.conf} module parameter control file.

To change this parameter, the Intel® MPSS host driver must be reloaded if it is currently running. Follow the procedure:

\begin{verbatim}
[host]# `service mpss unload
[host]# `service mpss start
\end{verbatim}

Registration caching is enabled by default.

\subsection*{9.3.5 Registration caching limit}

There is a per-node tunable limit on the maximum number of pinned pages per SCIF endpoint. This limit can only be modified by the root user.

- Set the maximum number of pinned pages by writing to the coprocessor’s \texttt{/proc/scif/reg\_cache\_limit} node:

\begin{verbatim}
[host]# echo \texttt{<limit>} > /proc/scif/reg\_cache\_limit
\end{verbatim}

where \texttt{<limit>} is the decimal number of 4K pages.

- To disable caching at runtime, set the \texttt{<limit>} to 0 on each node.

\subsection*{9.3.6 Huge page support}

SCIF has support for Huge Pages. Huge Pages should not to be confused with Transparent Huge Pages (THP); SCIF support of THP is always enabled.

- Huge Page support is controlled by the \textit{\texttt{huge\_page}} parameter in the host’s \texttt{/etc/modprobe.d/mic.conf} module parameter control file.

- If the host driver is loaded it must be reloaded. Follow this procedure:

\begin{verbatim}
[host]# `service mpss unload
[host]# `service mpss start
\end{verbatim}

Huge Page support is enabled by default.

\section*{9.4 COI configuration}

\subsection*{9.4.1 COI offload user options}

The \texttt{coi\_daemon} on a coprocessor spawns processes on behalf of client processes on the host processor.

\subsubsection*{9.4.1.1 Ownership modes}

The \texttt{coi\_daemon} has several options for assigning ownership of these COI processes.
9.4.1.1.1 Micuser ownership

When operating in micuser mode, each COI process spawned by the coi_daemon is owned by user micuser.

9.4.1.1.2 _authorized user ownership

When operating in _Authorized mode, each COI process spawned by the coi_daemon is owned by same user as the corresponding host client process. Authentication of user credentials occurs using an .mpsscokie file located in the user’s home directory. The cookie is created and managed by the host’s mpss daemon.

9.4.1.1.3 _dynamic user ownership

When operating in _Dynamic mode, each COI process spawned by the coi_daemon is owned by a new, unique user created by the coi_daemon. Files and directories created by such a process cannot be accessed by other COI processes. This effectively isolates all COI Processes from each other for better security.

Note: the _Dynamic mode will be removed in future release.

9.4.1.2 Configuring the ownership mode

The Ownership mode is configured by the presence of the parameter:

coiparams='--coiuser=<mode>'

in one of the following files in the coprocessor file system:

/etc/init.d/coi
/etc/coi.conf
/etc/sysconfig/coi.conf

and where <mode> is one of micuser, _Authorized, or _Dynamic.

micuser ownership mode is configured, by default, in the coprocessor’s /etc/init.d/coi file:

coiparams='--coiuser=micuser' #default parameters at boot

When /etc/coi.conf contains the coiparams parameter, it takes precedence over /etc/init.d/coi. When /etc/sysconfig/coi.conf contains the coiparams parameter, it takes precedence over /etc/coi.conf and /etc/init.d/coi.

A change to the ownership mode only occurs when the coi daemon is restarted:

[micN]# /etc/init.d/coi restart

Note: Changes to files that reside in coprocessor memory are not retained when the coprocessor is shut down or restarted. Refer to Section 7 for information on preserving changes to the coprocessor file system.
Alternatively, the --coiuser option can be passed to the coi_daemon when it is started:

```
[micN]$ coi_daemon --coiuser=<mode>
```

### 9.4.1.3 Example

The following configures for the _Authenticated user mode_, overriding whatever is configured in `/etc/init.d/coi` and `/etc/coi.conf`:

```
[micN]$ echo coiparams='--coiuser=_Authorized' > /
/etc/sysconfig/coi.conf
[micN]$ /etc/init.d/coi restart
```

For detailed information about the --coiuser and other coi_daemon parameters, run the coi_daemon on the coprocessor with the --help option:

```
[micN]$ coi_daemon --help
```

### 9.5 Virtual console configuration and access

On a SLES* host, minicom prompts for a username and password when logging in to the coprocessor. Use micctrl --passwd=<user> to set the password for a user before using the virtual console on minicom.

The virtual console devices are `/dev/ttyMICN` for the each coprocessor.

To configure minicom for virtual console access, perform the following instructions for each coprocessor:

1. Start minicom:

   ```
   [host]$ minicom -s
   ```

2. Select "Serial Port Setup"

   a. Choose option: A - _Serial Device_
   b. Edit Serial Device to `/dev/ttyMIC0`
   c. Hit <Enter> twice.

3. Select "Modem and dialing"

   a. Choose option: A - _Init string_
   b. Erase the entire line
   c. Press <Enter> twice

4. Select "Save setup as.."

   a. Enter the preferred name, for example: mic0
b. Press <enter>

5. Select “Exit from minicom”

Each coprocessor should have its own configuration name.

To open the virtual console for a coprocessor:

    [host]# minicom <configname>

where <configname> is the name entered in step 4.

To exit minicom, enter: <CTRL+a> <x> <Enter>

9.6  Virtio block device configuration and use.

The virtio block device (virtblk) uses the Linux* virtio data transfer mechanism to implement a block device on the coprocessor. The virtblk device stores data on the host, and therefore can be persistent like a hard or solid-state disk mounted on the coprocessor.

The virtio block device can be one of the following:

- A regular file, for instance: /srv/my_k1om_filesys, or
- A Logical Volume Manager (LVM) volume, or
- A physical device such as /dev/sda4

The block device to be used is communicated to the mic.ko driver by writing its path to the /sys/class/mic/micN/virtblk_file sysfs node after coprocessor micN has been booted. The virtblk driver supports only one virtio block device file on the host at any time. Once a virtio block device file is specified by writing to /sys/class/mic/micN/virtblk_file, it cannot be changed until the coprocessor is rebooted. To use multiple virtio block devices, create multiple partitions in a virtio block device file. Those partitions are referenced as /dev/vda1, /dev/vda2.

If a virtio block device file is not assigned, then unloading the Intel® MPSS host driver will trigger the message "request comes in while coprocessor side driver is not loaded yet. Ignore" in dmesg and /var/log/messages.

If the coprocessor side driver, mic_virtblk, is loaded without assigning a virtio block device file, the error message "Have set virtblk file?" will be displayed in dmesg and /var/log/messages.

9.6.1 Using a virtio block device as an ext2 file system

1. Host side:

   Identify the file or block device on which the virtblk file system will reside.

   [host]# echo <path_to_dev> > /sys/class/mic/micN/virtblk_file
2. Coprocessor side:
   a) Load the *virtblk* driver.
      
      [micN]# modprobe mic_virtblk
   
   b) Create ext2 file system on *virtblk* and mount it on /mnt/vda.
      
      [micN]# mkdir -p /mnt/vda
      [micN]# mkfs.ext2 /dev/vda
      [micN]# mount -t ext2 /dev/vda /mnt/vda
      
      You can now access /mnt/vda as a file structured device.

9.6.2 Use a *virtblk* device as a swap device file system

1. Host side:
   
   [host]# echo <path_to_dev> > /sys/class/mic/micN/virtblk_file

2. Coprocessor side:
   a) Load the *virtblk* driver:
      
      [micN]# modprobe mic_virtblk
   
   b) Assign a swap device and confirm:
      
      [micN]# swapon /dev/vda
      [micN]# cat /proc/swaps
   
   c) You can now use /dev/vda as a swap device.
A Intel® MPSS configuration parameters

This appendix describes the parameters in Intel® MPSS configuration files. The Parameter Syntax in the following sections sometimes extends to more than one line. However, each parameter in an actual software stack configuration file must be free of NewLines.

The first line of each parameter description is either:

Parameter Syntax (default.conf):

or:

Parameter Syntax (micN.conf):

indicating whether the parameter is created by default in a default.conf or micN.conf configuration file.

The line following Initial Value: in the descriptions below is the parameter value initially set by micctrl --initdefaults. Not all parameters have an initial value set by micctrl --initdefaults.

Intel® configuration file text lines beginning with the # character are treated as comments.

A.1 Meta configuration

A.1.1 Configuration version

Parameter Syntax (micN.conf):

    Version <major number> <minor number>

Initial Value:

    Version 1 1

(At this writing.)

Description:

The Version parameter sets the coprocessor configuration file version. As new releases are produced, the version is used by the micctrl --initdefaults command to identify where to update configuration files. This parameter should NOT be manually edited.
A.1.2 Including other configuration files

Parameter Syntax (micN.conf):

Include <config_file_name>

Initial Value:

Include default.conf
Include "conf.d/.*.conf"

Description:

Each configuration file can include other configuration files. The Include parameter lists configuration file(s) to be included. The configuration file(s) to be included must be in /etc/mpss. The configuration parser processes each parameter sequentially. When the Include parameter is encountered, the included configuration file(s) are immediately processed. If a parameter is set multiple times, the last instance of the parameter setting will be applied.

Each Include parameter should identify a single file to be included.

By default, the /etc/mpss/default.conf file is included at the beginning of each micN.conf coprocessor specific file. This allows parameters in the coprocessor specific file to override any parameter set in default.conf.

The second entry in the micN.conf files is typically (and by default) the line:

Include "conf.d/.*.conf"

This is a special rule, specifying that any configuration file that is placed in the /etc/mpss/conf.d directory will be included.

A.2 Boot control

A.2.1 What to boot

Parameter Syntax (micN.conf):

OSimage <linux_kernel_image> <system_address_map_file>

Initial Value:

OSimage /usr/share/mpss/boot/bzImage-knightscorner
/usr/share/mpss/boot/System.map-knightscorner

Description:

The OSimage parameter specifies the coprocessor Linux* OS boot image and its associated system address map file.

OSimage may be changed using the micctrl --osimage command or by editing this parameter directly.
A.2.2 When to boot

Parameter Syntax (micN.conf):

BootOnStart (Enabled|Disabled)

Initial Value:

BootOnStart Enabled

Description:

The BootOnStart parameter controls whether the coprocessor is booted when the mpss service starts. If set to Enabled, the mpssd daemon will attempt to boot the coprocessors when `service mpss start` is called.

BootOnStart may be changed using the `micctrl --autoboot` command or by editing this parameter directly.

A.3 Kernel configuration

These parameters influence or control the execution of the coprocessor Linux* kernel through values passed to the kernel in the startup command line.

A.3.1 ExtraCommandLine

Parameter Syntax (default.conf):

ExtraCommandLine "<commands>"

Initial Value:

ExtraCommandLine "highres=off"

Description:

The ExtraCommandLine parameter specifies additional kernel command line parameters to be passed to the coprocessor kernel on boot.

ExtraCommandLine may be changed by editing the parameter directly.

A.3.2 Console device

Parameter Syntax (default.conf):

Console "<console device>"

Initial Value:

Console "hvc0"
Description:

Intel® MPSS software supports a PCIe bus virtual console driver. Its device node (hvc0) is the default value assigned to the Console parameter, and should not be changed.

A.3.3 Power management

Parameter Syntax (micN.conf):

```
PowerManagement
  "cpufreq_(on|off);corec6_(on|off);pc3_(on|off);pc6_(on|off)"
```

Initial Value:

```
PowerManagement "cpufreq_on;corec6_off;pc3_on;pc6_on"
```

Description:

The PowerManagement parameter is a string of four attributes passed directly to the kernel command line for the coprocessor’s power management driver. The mpssd daemon and micctrl utility do not validate any of the parameters in this string or its format.

PowerManagement may be changed using the micctrl --pm command or by editing this parameter directly.

Coprocessor power states are described in the Intel® Xeon Phi™ Coprocessor Datasheet

Note: It is recommended to use the default power management settings unless directed by an Intel® representative to change them.

A.3.4 ShutdownTimeout

Parameter Syntax (default.conf):

```
ShutdownTimeout <value>
```

Initial Value:

```
ShutdownTimeout 300
```

Description:

Setting value to a positive integer specifies the maximum number of seconds to wait for the coprocessor to shut down. If shut down time exceeds the value, the coprocessor is reset.

Setting value to any negative value indicates to wait indefinitely for the coprocessor to shut down.

Setting value to zero indicates to reset the coprocessor without waiting for it to shut down.

ShutdownTimeout can be changed by editing the parameter directly.
A.3.5 CrashDump

*Parameter Syntax (default.conf):*

```
CrashDump <dirname> <limit>
```

*Initial Value:*

```
CrashDump /var/crash/mic 16
```

*Description:*

The *CrashDump* parameter specifies the host directory, `<dirname>`, in which to place coprocessor crash dump files, and the maximum size, `<limit>`, in gigabytes of such files.

*CrashDump* can be changed by editing the parameter directly.

A.3.6 Cgroup

*Parameter Syntax (micN.conf):*

```
Cgroup [memory=(disabled|enabled)]
```

*Initial Value:*

```
Cgroup memory=disabled
```

*Description:*

The *Cgroup* parameter configures cgroups categories. Their configuration is currently limited to controlling the status of the memory cgroup.

The memory cgroup is disabled by default. Enabling cgroup memory support may reduce performance.

*Cgroup* may be changed using the *micctrl --cgroup* command or by editing the parameter directly.

A.3.7 VerboseLogging

*Parameter Syntax (micN.conf):*

```
VerboseLogging (Enabled|Disabled)
```

*Initial Value:*

```
VerboseLogging Disabled
```

*Description:*

The *VerboseLogging* parameter specifies whether the *quiet* kernel command line parameter is passed to the coprocessor on boot. The *quiet* kernel parameter suppresses most kernel messages during kernel boot. *VerboseLogging* is disabled by default. Enabling *VerboseLogging* will increase boot times.
VerboseLogging may be changed by editing the parameter directly.

**Note:** This parameter may be deprecated in future releases.

### A.4 File system configuration parameters

#### A.4.1 RootDevice

**Parameter Syntax (micN.conf):**

```
RootDevice (Ramfs|StaticRamfs) <ramfs_location>
RootDevice NFS <share>
RootDevice SplitNFS <share> <usr_share>
```

**Initial Value:**

RootDevice Ramfs /var/mpss/mic0.image.gz

**Description:**

The **RootDevice** parameter defines the type of root device to mount. Supported types are **RamFS**, **StaticRamFS**, **NFS**, and **SplitNFS**.

The **RamFS** type builds a compressed cpio ram disk image when a request to boot is received. `<ramfs_location>` is the directory path and file name of the resulting ram disk image. The image is used as the contents to be loaded into the root tmpfs file system.

The **StaticRamFS** type causes the compressed cpio image `<ramfs_location>` to be used as the contents of the root file system for the booting coprocessor. The **StaticRamFS** boot will fail if the image file is not already present at `<ramfs_location>`.

The static ramfs image may have been previously created by booting with **RootDevice** set to **RamFS**. Optionally, when **RootDevice** is **StaticRamFS**, the `micctrl --updateramfs` command causes a compressed cpio image to be built and placed at the `<ramfs_location>` of the **StaticRamFS** parameter. System administrators may also supply their own initial ram disk image.

The **NFS** type instructs the booting coprocessor to mount the NFS share specified by the `<share>` argument as the root file system. `<share>` must be a fully qualified NFS mount location with the format "server:location", for example `10.10.10.1:2/export/mic0`.

The **SplitNFS** type is the same as **NFS** except it also provides a separate NFS share at `<usr_share>` to mount as the `/usr` directory on the coprocessor.

**RootDevice** may be changed using the `micctrl --rootdev` command or by editing the parameter directly.

#### A.4.2 File locations

The `mpssd` daemon and `micctrl` command use the **Base**, **CommonDir**, **Overlay**, and **MicDir** parameters to locate the files which will be placed in the coprocessor's final root disk image. Out of the four parameters, only the **Overlay** is allowed to be specified
multiple times. These parameters collectively specify all the files from which a root file system cpio image is to be built.

A.4.2.1 Base

Parameter Syntax (micN.conf):

Base (CPIO|DIR) <target>

Initial Value:

Base CPIO /usr/share/mpss/boot/initramfs-knightscorner.cpio.gz

Description:

The Base parameter specifies the file system hierarchy over which other hierarchies are overlaid to produce the initial file system of a coprocessor. When the Base type is CPIO, <target> is interpreted as the file name of a CPIO file system archive. When the Base type is DIR, <target> is interpreted as the root of an expanded (non-archived) file system.

Base may be changed using the micctrl --base command or by editing the parameter directly.

A.4.2.2 CommonDir

Parameter Syntax (default.conf):

CommonDir <source> <target>

Initial Value:

CommonDir /var/mpss/common

Description:

The CommonDir parameter defines a directory at <source> whose contents overlay the Base file system at /. Thus if CommonDir is /var/mpss/common and there is a file /var/mpss/common/foo/bar, then that file will be found as /foo/bar in the resulting file system.

Intel® MPSS installation does not create or populate the CommonDir directory. It is typically created by the micctrl --initdefaults command. Files that are added to this directory are maintained between updates to the Intel® MPSS installation.

<target> is a deprecated argument, which will be ignored. If present when micctrl --resetdefaults is executed, the <target> argument will be removed.

CommonDir may be changed using the micctrl --commondir command or by editing the parameter directly.
A.4.2.3 MicDir

Parameter Syntax (micN.conf):

MicDir <location>

Initial Value:

MicDir /var/mpss/micN

Description:

The MicDir parameter defines a directory at <location> whose contents overlay the CommonDir file system at / to create a file system unique to each coprocessor. Intel® MPSS installation does not create or populate the MicDir directory. It is typically created by the micctrl --initdefaults command. Files that are added to this directory are maintained between updates to the Intel® MPSS installation.

MicDir may be changed using the micctrl --micdir command or editing the parameter directly.

Note: In some previous Intel® MPSS versions, MicDir took a <descriptor file> parameter:

MicDir <location> <descriptor file>

The <descriptor file> parameter used to identify a file that described where files in the directory subtree at <location> were to be placed in the coprocessor’s file system, and the permissions of those files. The <descriptor file> parameter to MicDir has been deprecated. New configuration files generated with the micctrl --initdefaults command do not include it. If the micctrl --resetdefaults command is executed, the <descriptor file> argument will be removed wherever it is found.

A.4.2.4 Overlay

Parameter Syntax (micN.conf):

Overlay (Filelist|Simple|File) <source> <target> (on|off)
Overlay RPM <source> (on|off)

Initial Value:

<None>

Description:

The Overlay parameter specifies a file or set of files that are to be added to the initial file system, overlaying the Base, CommonDir, and MicDir specified directory hierarchies. There can be multiple Overlay parameters. If the Overlay state value is off, the parameter is ignored. The Overlay parameter is obeyed if the state value is on.

Overlay File overlays the file <source> onto the initial file system image at <target>. Directory and file ownership and permissions are preserved.
**Intel® MPSS configuration parameters**

*Overlay Simple* overlays the file system hierarchy at `<source>` onto the initial file system image at `<target>`. Directory and file ownership and permissions are preserved.

*Overlay RPM* copies the `<source>` file to a special `/RPMs-to-install` directory in the initial file system. During the coprocessor boot process, the *init* program will attempt to install any RPMs which it finds in that directory. Other types of files are ignored.

*Overlay Filelist* overlays files in the directory `<source>` onto the initial file system image based on specifications in the `<target>` file. Use of *Overlay Filelist* is deprecated.

*Overlay* may be changed using the *micctrl --overlay* command or editing the parameter directly.

**Note:** Do **not** overlay `$MPSS38_K1OM`. That is, do **not** define a parameter similar to:

```
Overlay RPM $MPSS38_K1OM
```

Doing so will cause *micctrl* to attempt to upload and install all the RPMs in the `$MPSS38_K1OM`, and will likely result in the coprocessor running out of memory or hanging.

### A.4.3 Intel® MPSS RPM location

**Parameter Syntax (micN.conf):**

```
K1omRpms <location>
```

**Initial Value:**

```
<None>
```

**Description:**

The implementation of some *micctrl* commands, specifically those which configure for use of *LDAP* and *NIS* services, needs to know where to find the set of RPM files that it needs to complete installation of *LDAP* and *NIS* in the coprocessor file system. The *K1omRpms* parameter should point to such a `<location>`. This parameter is not defined by default. In general, it can be set to the directory which we refer to symbolically as `$MPSS38_K1OM`.

*K1omRpms* may be changed using the *micctrl --rpmdir* command or by editing the parameter directly.
A.5 Network configuration

A.5.1 Host name assignment

Parameter Syntax (micN.conf):

Hostname <name>

Initial Value:

<host name>-micN.<domain>

or

<host name>-micN

where <host name> is the “short” hostname of the host platform, as returned by calling hostname -s, micN is the coprocessor name, and <domain> is the host’s domain name as return by hostname -d.

Description:

The Hostname parameter specifies the host name value to be inserted in the hostname.conf file of coprocessor micN.

HostName may be changed by editing the parameter directly.

A.5.2 MAC address assignment

Parameter Syntax (micN.conf):

MacAddrs (Serial|Random|<host MAC>:<card MAC>)

Initial Value:

MacAddrs Serial

Description:

MAC addresses must be generated for the virtual network interfaces of the host and coprocessors. However, as a prerequisite, both ends of the virtual network need to have MAC addresses assigned.

By default, MAC addresses are generated based on the serial number of the coprocessor. Some older coprocessors do not have a usable serial number; in that case the MAC address is generated randomly.

The least significant bit is set in MAC addresses generated for host endpoints, and clear in MAC addresses generated for coprocessor endpoints. In addition, the top three octets of generated MAC addresses have the IEEE assigned value 4C:79:BA to enable identification of coprocessor interfaces.

The system administrator may override the default Serial behavior with the MacAddrs configuration parameter. For MacAddrs Random, random addresses are generated.
For **MacAddrs** `<host MAC>:<card MAC>`, the specified MAC addresses are statically assigned to the host and coprocessor network endpoints of micN.

**MacAddrs** may be changed using the `micctrl --mac` command or by editing the parameter directly.

### A.5.3 Static pair topology

**Parameter Syntax (micN.conf):**

```
Network class=StaticPair micip=<cardIP> hostip=<hostIP>
mtu=<mtu size> netbits=<netbits> modhost=(yes|no)
modcard=(yes|no|<path_to_file>)
```

**Initial Value:**

```
Network class=StaticPair micip=172.31.<N+1>.1
hostip=172.31.<N+1>.254 mtu=64512 netbits=24 modhost=yes
modcard=yes
```

for coprocessor micN.

**Description:**

In the static pair network topology, every coprocessor is assigned to a separate subnet known only to the host.

`<cardIP>` and `<hostIP>` are the IP addresses of the coprocessor and host endpoints. They must each be a fully qualified IP address, and the first three quads of the address must match.

The `mtu` parameter specifies the packet size to use over the virtual network connection.

The `netbits` argument specifies the number of high order bits that are set in the Netmask. The `<hostIP>` and `<cardIP>` must be identical over the high order `<netbits>` bits. The default value is 24, defining a netmask of 255.255.255.0. For a static pair configuration it should never be necessary to change this parameter. It is up to the system administrator to correctly route the virtual Ethernet nodes to the external network or each other.

If `modhost` is set to `yes`, the coprocessor's 'IPaddress hostname' pair is appended to the contents of the host's `/etc/hosts` file with the comment '#Generated-by-micctrl'. If `modhost` is set to `no` the entry matching the coprocessor's IP address with the comment '#Generated-by-micctrl' will be removed from the host's `/etc/hosts` file.

If `modcard` is set to `yes`, an `/etc/hosts` file is created in the coprocessors file system, containing the 'IPaddress hostname' pair of both the host (bridge) and the coprocessor. If `modcard` is set to `no`, the `/etc/hosts` will not be created. User may also provide a path to a file which content will be copied to `<vardir>/etc/hosts`.

**Note:** The `modhost` and `modcard` options, if present, override the deprecated `hosts` parameter. The `hosts=(yes|no)` option may still be used; setting is equivalent to setting `modhost` and `modcard` with the specified `(yes|no|<path_to_file>)` value.
Although the static pair network configuration can be changed by editing `micN.conf` and `default.conf` configuration files, the recommended method of changing the network configuration is to use the `micctrl --network` command (see Appendix B.4.5 for details). Specifically, the `micctrl --network` command will edit configuration files as needed to remove the current network configuration before implementing the new configuration.

Linux* networking supports routing a static pair to the external network and or to another static pair. It is the responsibility of the system administrator to configure such routing.

### A.5.4 Internal bridge topology

**Parameter Syntax:**

```
Bridge <name> Internal <bridgeIP> <netbits> <mtu>
Network class=StaticBridge bridge=<name> micip=<cardIP>
modhost=(yes|no) modcard=(yes|no|<path_to_file>)
```

**Initial Value:**

<None>

**Description:**

Linux* provides a mechanism for bridging network devices to a common network. The term "internal bridge", refers to a configuration in which the host and one or more coprocessors on that host are connected through a bridge.

The internal bridge configuration is specified by a pair of parameters: a `Bridge` parameter in the `default.conf` file to specify the bridge information, and a `Network` parameter, in the `micN.conf` file of each coprocessor to be bridged, which specifies the bridge to which the coprocessor is connected and other information.

The same bridge name, `<name>`, must be given to both the `Bridge` and `Network` parameters.

`<bridgeIP>` and `<cardIP>` are the IP addresses of the bridge and coprocessor endpoints respectively. The `<netbits>` argument to `Bridge` specifies the number of high order bits that are set in the Netmask. The `<bridgeIP>` and `<cardIP>` must be identical over the high order `<netbits>` bits. For example, if `<netbits>` is 24, then the Netmask is 255.255.255.0, and IP addresses must be identical over the first three quads.

The `<mtu>` argument to `Bridge` specifies the packet size to use over the virtual network connection. The value of 64k has been shown to provide the highest network performance.

If `modhost` is set to `yes`, the coprocessor's 'IPaddress hostname' pair is appended to the host's `/etc/hosts` file with the comment '#Generated-by-micctrl'. If `modhost` is set to `no` the entry matching the coprocessor's IP address with the comment '#Generated-by-micctrl' will be removed from the host's `/etc/hosts` file.

If `modcard` is set to `yes`, an `/etc/hosts` file is created in the coprocessors file system, containing the 'IPaddress hostname' pair of both the host (bridge) and the coprocessor. If `modcard` is set to `no`, the `/etc/hosts` will not be created. User may also provide a path to a file which content will be copied to `<vardir>/etc/hosts`. 
**Note:** The *modhost* and *modcard* options, if present, override the deprecated *hosts* parameter. The *hosts*=(yes|no) option may still be used; setting is equivalent to setting *modhost* and *modcard* with the specified (yes|no|<path_to_file>) value.

The resulting configuration files will use the Bridge parameters for <mtu> and <netbits> values for the coprocessor endpoints.

The recommended method of changing the Bridge and Network parameters is to use the *micctrl --network* command (see Appendix B.4.5.4 for details), rather than by directly editing. The --network command evaluate the current network configuration and can remove it before creating the new one. In either case, all the network control files will be created when the operation is done.

### A.5.5 External bridge topology

**Parameter Syntax:**

Bridge <name> External <bridgeIP> <netbits> <mtu>

Network class=StaticBridge bridge=<name> micip=<cardIP>
[mtu=<mtu size>] [netbits=<netbits>] modhost=(yes|no)
modcard=(yes|no|<path_to_file>)
Bridge <name> External dhcp
Network class=Bridge bridge=<name>

**Initial Value:**

<None>

**Description:**

The Linux* bridging mechanism can bridge the coprocessor virtual connections to a physical Ethernet device. In this topology, the virtual network interfaces become configurable to the wider subnet.

The external bridge configuration is specified by a pair of parameters: a Bridge parameter in the default.conf file to specify the bridge information, and a Network parameter, in the micN.conf file of each coprocessor to be bridged, which specifies the bridge to which the coprocessor is connected.

The same bridge name, <name>, must be used in both the Bridge and Network parameters.

IP addresses of the bridge and coprocessor endpoints can be statically assigned or configured for DHCP dynamic assignment.

The bridge IP address is assigned statically by specifying the <bridgeIP> argument to Bridge. The <mtu> argument to Bridge specifies the packet size to use over the virtual network connection. The default value is 1500 bytes to match default physical network settings. If attaching to a pre-existing external bridge configuration, the specified mtu value must match the setting in the system configuration file. For example, if, on a RHEL* based host, the /etc/sysconfig/network-scripts/ifcfg-br0 file contains the line "MTU=9000", then the MTU field must be set to 9000 to match. The <netbits> argument to Bridge specifies the number of high order bits that are set in the Netmask. It must be a value between 8 and 24. By default it is set to 24 by default and will rarely need to be changed.
Coprocessor IP address assignment is static when the Network class=StaticBridge; the coprocessor IP address is specified by <cardIP>. The <bridgeIP> and <cardIP> must be identical over the high order <netbits> bits. The resulting configuration will use the bridge’s <mtu> and <netbits> values to ensure they match.

If modhost is set to yes, the coprocessor’s 'IPaddress hostname' pair is appended to the host’s /etc/hosts file with the comment '#Generated-by-micctrl'. If modhost is set to no the entry matching the coprocessor’s IP address with the comment '#Generated-by-micctrl' will be removed from the host's /etc/hosts file.

If modcard is set to yes, an /etc/hosts file is created in the coprocessors file system, containing the 'IPaddress hostname' pair of both the host (bridge) and the coprocessor. If modcard is set to no, the /etc/hosts will not be created. User may also provide a path to a file which content will be copied to <vardir>/etc/hosts.

**Note:** The modhost and modcard options, if present, override the deprecated hosts parameter. The hosts=(yes|no) option may still be used; setting is equivalent to setting modhost and modcard with the specified (yes|no|<path_to_file>) value.

The bridge IP address is configured for dynamic assignment by including value dhcp instead of a static IP address in the Bridge parameter. The DHCP server will also assign the mtu and Netmask values.

Coprocessor IP address assignment is dynamic by DHCP when Network class=Bridge.

The modhost and modcard parameters are not required because it is assumed the IP address for each coprocessor will be retrievable from a name server on the network.

If the corresponding bridge networking configuration file (ex: ifcfg-br0) does not exist then this parameter will cause it to be generated.

The Intel® configuration will not generate or modify the physical interface file to attach the physical network to the bridge. The system administrator must perform this step. For example, on a RHEL* host, a file /etc/sysconfig/network-scripts/ifcfg-eth0 to link the eth0 interface to bridge br0 might have the following contents:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

On SLES* host platforms, the physical port name must be added to the BRIDGE_PORTS entry in the /etc/sysconfig/networks/ifcfg-br0 configuration file, for example:

```
BRIDGE_PORTS=’eth0 mic0 mic1’
```

The recommended method of changing the Bridge and Network parameters is to use the micctrl --network command (see Appendix B.4.5.5 for details), rather than by directly editing. The --network command evaluate the current network configuration and can remove it before creating the new one. In either case, all the network control files will be created when the operation is done.
A.6 Deprecated configuration parameters

A.6.1 User access

Parameter Syntax (micN.conf):

```
UserAuthentication None
UserAuthentication Local <low uid> <high uid>
```

Description:

The UserAuthentication parameter has been removed. Refer to the sections on micctrl specification of users for the coprocessors for configuration user access.

A.6.2 Service startup

Note: This parameter is still functional but there are no longer default services using it. It may be fully deprecated and removed in the future.

Parameter Syntax (micN.conf):

```
Service <name> <start> <stop> <state>
```

Description:

During boot, the embedded Linux* OS on the coprocessor executes the script files in the `/etc/rc5.d` directory. These entries are links to the actual script files in the `/etc/init.d` directory. The links are named with the standard Linux* custom starting with an 'S' for start or 'K' for stop followed by the position parameter and then the file name from the init.d directory. The position parameter is a number from 01 to 99 establishing the order in which the scripts are executed.

The Intel® MPSS installs several pieces of software with various service scripts. The system administration may not want all of them to start at boot. To support this functionality, the configuration files specify the creation of the files in `/etc/rc5.d` based on the Service configuration parameter. Each file in `/etc/init.d` will require a Service entry in the coprocessor configuration file.

The name argument is the name of the actual script found in the `/etc/init.d` directory.

The start argument defines the order the service start relevant to other scripts. It will be a value from 1 to 99. As an example the network interface must be initialized before the secure shell daemon can be started. The network script is assigned a start value of 21 and sshd is assigned 80.

The stop argument is the opposite of the start parameter and is generally set to 100 minus the start value. This will assure that on shutdown the secure shell daemon at 5 will shut down before the network is unconfigured at 79.

The state argument determines whether the links specifies an 'S' for start or 'K' for stop. It follows the chkconfig utility convention of on for start and off for stop.
B  The micctrl utility

The micctrl utility is a multi-purpose tool for the system administrator. It provides the following categories of functionality:

- Coprocessor state control – boot, shutdown and reset control while the mpssd daemon is running.
- Configuration file initialization and propagation of values.
- Helper functions for modifying configuration parameters.
- Helper functions for modifying the root file system directory or associated download image.

B.1  Micctrl command line format

The micctrl command line format is:

    micctrl GlobalOptions Command SubOptions Coprocessors

GlobalOptions is a space separated list of 0 or more of the global options documented in Appendix B.2. GlobalOptions list elements can appear in any order.

Command is one of the commands documented in Appendix B.4 through Appendix B.4.8.

SubOptions is a space separated list of 0 or more suboptions. A suboption can be one of the Global SubOptions described in Appendix B.3.1, or one of the Common SubOptions documented in Appendix B.3.2 or a command-specific suboption described in the documentation of the specified command. SubOptions list elements can appear in any order.

Coprocessors is a space separated list of 0 or more coprocessor identifiers of the form micN. Coprocessors list elements can appear in any order. If the Coprocessors list is empty, and the mic.ko driver is loaded, the command is applied to all discovered coprocessors. If the mic.ko driver is not loaded, then the Coprocessors list must be non-empty.

Note: Providing invalid coprocessors’ identifiers will return an error message or may cause the micctrl command to create unwanted configuration files.

For brevity, the command-specific syntax of each command in Appendix B.4 through Appendix B.4.8 does not include GlobalOptions, Coprocessors or Global SubOptions. For example, the syntax of the micctrl wait command is shown as:

    micctrl (-w|--wait)

rather than the full syntax:

    micctrl [(-d |--destdir=<destdir>) \  
     [(-c |--configdir=<confdir>) \  
     (-w|--wait) [(-h|--help)] [(-v|--vv|--v|-vvv|--v -v -v)] \ 
     Coprocessors]
Some aspects of network configuration are operating system dependent. By default, 
micctrl performs network configuration operations according to the operating system of 
the local host. The distrib suboption can be used to force micctrl to perform network 
configuration for a specified operating system. It is typically used with other options 
such as --destdir and --netdir when creating a configuration to be pushed to another 
system.

B.2 Global options

Global options are common to all micctrl commands.

B.2.1 --destdir, -d

Option Syntax:


[(-d |--destdir=)<destdir>]

Description:

The --destdir global option, if specified, overrides the current value of the implicit 
$DESTDIR variable.

We use the symbol $DESTDIR to indicate the directory path that micctrl prepends to all 
micctrl accesses of micctrl created files.

$DESTDIR is described in Section 4.1.3.1.

B.2.2 --configdir, -c

Option Syntax:


[(-c |--configdir=)<confdir>]

Description:

The --configdir global option, if specified, overrides the current value of the implicit 
$CONFIGDIR variable.

We use the symbol $CONFIGDIR to indicate the directory path at which micctrl creates 
Intel® MPSS-specific configuration files, specifically default.conf and micN.conf.

$CONFIGDIR is described in Section 4.1.3.2.

B.3 Suboptions

Some suboptions are unique to each micctrl command; these are described with each of 
the commands. Other suboptions are common to some or all commands.
B.3.1 Global suboptions

Global suboptions are common to all commands. For brevity, command syntax does not show these global suboptions. For example, the syntax of the `micctrl --status` command is shown as:

```
micctrl (-s|--status)
```

rather than:

```
micctrl (-s|--status) [(-h|--help)] \ 
[(-v|--vv|--vvv|--vv|--v|--v--v|--v--v)--v--v)]
```

B.3.1.1 Help

**Suboption Syntax:**

```
[(-h|--help)]
```

**Description:**

The `-h` suboption cause `micctrl` to ignore all other options and output help for the specified Command. For example to get help on the `micctrl --initdefaults`, use the `-h` option:

```
micctrl --initdefaults -h
```

B.3.1.2 Verbose output

**Suboption Syntax:**

```
[(-v|--vv|--v|--vvv|--vv|--v|--v)]
```

**Description:**

By default `micctrl` only outputs errors and warnings.

The `-v` suboption causes `micctrl` to output additional informational messages. The `-vv` or `-v -v` suboptions add notification of changes to all files `micctrl` is creating, deleting or changing. The `-vvv` or `-v -v -v` suboptions add notification of calls to the host's networking utilities, for instance: `ifup`.

B.3.2 Common suboptions

Some suboptions are common to several commands. For brevity, we define these suboptions once here. When an individual command supports a particular common suboption, the command syntax shows it, but does not include its description.

For example, the syntax of the `micctrl --rootdev` command is given as:

```
micctrl --rootdev=(RamFS|StaticRamFS) [--vardir=<vardir>] \ 
[(-t|--target=)<location>] [(-d|--delete)]
```
The description of the *micctrl* --rootdev command does not include a description of the --vardir suboption, which is common to several commands and is described in this section.

### B.3.2.1 --vardir

**Suboption Syntax:**

```
[--vardir=<vardir>]
```

**Description:**

The --vardir suboption, if specified, overrides the current value of the implicit $VARDIR variable.

We use the symbol $VARDIR to indicate the directory path at which the *micctrl* --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands create the common and micN overlay hierarchies, and at which the *micctrl* --rootdev command places a ramfs file system image or NFS file system hierarchy. By default $VARDIR is /var/mpss.

$VARDIR is described in Section 4.1.3.3.

### B.3.2.2 --srcdir

**Suboption Syntax:**

```
[--srcdir=<srcdir>]
```

**Description:**

The --srcdir suboption, if specified, overrides the current value of the implicit $SRCDIR variable.

We use the symbol $SRCDIR to indicate the directory path at which the *micctrl* --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands look for the coprocessor's Linux* kernel image and default file system image.

$SRCDIR is described in Section 4.1.3.4.

### B.3.2.3 --netdir, -n

**Suboption Syntax:**

```
[(-n | --netdir)=]<netdir>
```

**Description:**

The --netdir suboption, if specified, overrides the current value of the implicit $NETDIR variable.

We use the symbol $NETDIR to indicate the directory path at which the *micctrl* --initdefaults, --resetdefaults, --resetconfig, and --cleanconfig commands create and/or edit network control files.
$\text{NETDIR}$ is described in Section 4.1.3.5.

B.3.2.4  --distrib, -d

Suboption Syntax:

\[ [ (-d | --distrib=) (\text{redhat}|\text{suse}) ] \]

Description:

Some aspects of network configuration are operating system dependent. By default, \textit{micctrl} performs network configuration operations according to the operating system of the local host. The \textit{--distrib} suboption can be used to force \textit{micctrl} to perform network configuration for a specified operating system. It is typically used with other options such as \textit{--destdir} and \textit{--netdir} when creating a configuration to be pushed to another system.

B.3.2.5  --gw, -g

Suboption Syntax:

\[ [ (-g | --gw=) \langle \text{gateway} \rangle ] \]

Description:

The \textit{--gw} suboption sets the gateway of a coprocessor network interface. If not specified, the gateway of the local host is assigned to the network. The \textit{--gw} option is typically used with other options such as \textit{--destdir} and \textit{--netdir} when creating a configuration to be pushed to another system.

B.3.2.6  --users, -u

Suboption Syntax:

\[ [ (-u | --users=) (\text{none}|\text{overlay}|\text{merge}|\text{nochange}) ] \]

Description:

The \textit{--users} suboption controls creation and/or modification of the \texttt{/etc/passwd} and \texttt{/etc/shadow} files of each specified coprocessor. The \textit{MicDir} parameter specifies the directory in which these files are created and/or modified.

For \textit{--users=none}, the \texttt{/etc/passwd} and \texttt{/etc/shadow} files are deleted and recreated to include only the minimal set of users required by Linux*, which are the root, ssh, nobody, nfsnobody and micuser.

For \textit{--users=overlay}, the \texttt{/etc/passwd} and \texttt{/etc/shadow} files are deleted and recreated to include the users from the 'none' option and any regular users found in the \texttt{/etc/passwd} file of the host.

For \textit{--users=nochange}, behavior is as for \textit{--users=overlay} if no configuration exists for the specified coprocessor. Otherwise the \texttt{/etc/passwd} and \texttt{/etc/shadow} files are unchanged.
For --users=merge, any users in the host's /etc/passwd file but not in the coprocessor's /etc/passwd file are added to the coprocessor's /etc/passwd and /etc/shadow files.

If the --users suboption is not given, behavior is as for --users=nochange.

B.3.2.7 --pass, -a

Suboption Syntax:

\[ (-a | --pass=) (none | shadow) \]

Description:

The --pass suboption selects the policy for copying passwords from the host's /etc/shadow file to the specified coprocessor's /etc/shadow file. For --users=none and --users=overlay, the policy is applied to all users in the newly created /etc/shadow file. For --users=merge, the policy is only applied to users that are added to the /etc/shadow file.

For --pass=shadow, the host's /etc/shadow file is parsed and values of the affected users are written to the coprocessor's /etc/shadow file. It should be noted that --pass=shadow is disabled on SLES* host systems; SLES* uses Blow Fish encryption, which is not supported on the coprocessor.

For --pass=none, the passwords in the coprocessor's /etc/shadow file are set to the '*'.

If the --pass suboption is not given, behavior is as for --pass=shadow on a RHEL* based host, and as for --pass=none on a SLES* based host.

B.3.2.8 --modhost, -c

Suboption Syntax:

\[ (-c | --modhost=) (yes | no) \]

Description:

For --modhost=yes, the coprocessor's 'IPaddress hostname' pair is appended to the contents of the host's /etc/host file with the comment '#Generated-by-micctrl'.

For --modhost=no, the host's /etc/hosts file is unchanged.

Note: The --modhost suboption behaves as for --modcard=no if the configuration files already exist. In this case, use the micctrl --modbridge or --network command to change the host's /etc/hosts.
B.3.2.9  --modcard, -e

Suboption Syntax:

[ (-e | --modcard=) (yes|no|<path_to_file>) ]

Description:

For --modcard=yes, an /etc/hosts file is created and populated in directory defined by the MicDir parameter of each specified coprocessor.

For --modcard=no, /etc/hosts files are not created.

For --modcard=<path_to_file> the content of a specified file will be copied to <vardir>/etc/hosts

If the --modcard option is not given, behavior is as for --modcard=yes.

The --modcard suboption behaves as for --modcard=no if the configuration files already exist. In this case, use the micctrl --modbridge or --network command to change the host's /etc/hosts.

B.3.2.10  --nocreate

Suboption Syntax:

[--nocreate ]

Description:

By default, a home directory is created in the coprocessor file system for each user in a coprocessor's /etc/passwd and /etc/shadow files. The --nocreate suboption disables the creation of such home directories. Doing so can reduce ram file system memory usage when LDAP home directory auto mount is enabled or the /home directory is NFS mounted.

B.3.2.11  --pm, -p

Suboption Syntax:

[ (-p | --pm=) (default|defaultb) ]

Description:

The --pm suboption modifies the PowerManagement parameter of each specified coprocessor.

For --pm=default, the PowerManagement parameter of each specified coprocessor is reset to default setting for the coprocessor's stepping. If the stepping cannot be determined, the power management parameters are set to the default for the C stepping.

For --pm=defaultb, the PowerManagement parameter of each specified coprocessor is reset to the default setting for the coprocessor B stepping.

If the --pm suboption is not given, behavior is as for --pm=default.
B.4 Micctrl command descriptions

This section describes each of the micctrl commands.

The $DESTDIR, $CONFIGDIR, $VARDIR, $SRCDIR and $NETDIR directory path modifiers can alter the default directory paths of files which micctrl accesses. For brevity, the following description assumes default values for these intrinsic variables. See Section 4.1.3 for details.

B.4.1 Coprocessor state control

Several commands are available for controlling the state of coprocessors.

B.4.1.1 Booting the coprocessor

Command Syntax:

```
micctrl (-b|--boot) [(-w|--wait) [(-t|--timeout=)<timeout>]]
```

Description:

The micctrl --boot command requests that the specified coprocessors be booted. micctrl uses configuration parameters to prepare and boot coprocessors. The process depends on the root device type specified by the RootDevice parameter. Refer to Appendices A.4.1 and B.4.3 for details on root device types. Refer to Section 4.1.5 for a detailed description of the boot process.

By default, control returns before booting is complete. If the --wait suboption is specified, control returns after booting is complete, or after a timeout period, which ever is first.

If the --timeout suboption is specified, the timeout period is <timeout> seconds. If not specified, timeout period defaults to 300 seconds.

B.4.1.2 Shutting down the coprocessor

Command Syntax:

```
micctrl (-S|--shutdown) [(-w|--wait) \n [(-t|--timeout=)<timeout>]]
```

Description:

The micctrl --shutdown command requests that the specified coprocessors be shut down.

This command brings down the specified coprocessors in a safe way, and is equivalent to executing the Linux* shutdown command on each of the specified coprocessors.

By default, control returns before shutting down is complete. If the --wait suboption is specified, control returns after shutting down is complete, or after a timeout period, which ever is first.
If the --timeout suboption is specified, the timeout period is <timeout> seconds. If not specified, timeout period defaults to 300 seconds.

**B.4.1.3 Rebooting the coprocessor**

**Command Syntax:**

```
micctrl (-R|--reboot) [(-w|--wait) \ 
[(-t |--timeout=)<timeout>]]
```

**Description:**

The `micctrl --reboot` command requests that the specified coprocessors be rebooted. This command effectively performs the `micctrl --shutdown` followed by the `micctrl --boot` command.

By default, control returns before rebooting is complete. If the --wait suboption is specified, control returns after rebooting is complete, or after a timeout period, whichever is first.

If the --timeout suboption is specified, the timeout period is <timeout> seconds. If not specified, timeout period defaults to 300 seconds.

**B.4.1.4 Resetting the coprocessor**

**Command Syntax:**

```
micctrl (-r|--reset) [(-f|--force)] [(-i|--ignore)] \ 
[(-w|--wait) [(-t |--timeout=)<timeout>]]
```

**Description:**

The `micctrl --reset` command requests that the specified coprocessors be reset. The coprocessors can be in any state.

The --force suboption forces the coprocessors to go through the reset process. Normally the driver will not reset a coprocessor that is already in the reset or 'ready' state and `micctrl` will return an error.

The --ignore suboption prevents `micctrl` from returning an error if a coprocessor is in the reset or ready state.

By default, control returns before resetting is complete. If the --wait suboption is specified, control returns after resetting is complete, or after a timeout period, whichever is first.

If the --timeout suboption is specified, the timeout period is <timeout> seconds. If not specified, timeout period defaults to 300 seconds.

**Note:** Performing a reset may result in the loss of file data that has not been flushed to a remote file system. It is therefore recommended to perform a shutdown when this would not be desired.
B.4.1.5  Waiting for the coprocessor state change

Command Syntax:

```
micctrl (-w|--wait) [(-t|--timeout=<timeout>)]
```

Description:

The `micctrl --wait` command returns after the previous state change command is complete or after a timeout period, which ever is first.

If the `--timeout` suboption is specified, the timeout period is `<timeout>` seconds. If not specified, the timeout period defaults to 300 seconds.

B.4.1.6  Coprocessor status

Command Syntax:

```
micctrl (-s|--status)
```

Description:

The `micctrl --status` command displays the status of the specified coprocessors. If the status is online or booting it also displays the name of the associated boot image.

B.4.2  Initializing and propagating the configuration

This section discusses the `micctrl` command options for initializing, propagating, resetting, and cleaning configuration parameters.

B.4.2.1  Initializing the configuration files

Command Syntax:

```
micctrl --initdefaults [--vardir=<vardir>] [--srcdir=<srcdir>] \  
[(-d|--netdir=<netdir>)] \  
[(-u|--users=)(none|overlay|merge|nochange)] \  
[(-a|--pass=)(none|shadow)] [--nocreate] \  
[(-c|--modhost=(yes|no)) [(-e|--modcard=)(yes|no)<path_to_file>]]
```

Description:

The Intel® MPSS distribution does not include the software stack configuration files (For Instance: `default.conf`, `micN.conf`) and overlay hierarchies (For Instance: `/var/mpss/micN`). The `micctrl --initdefaults` command creates those files with default values, and can ensure that they are complete.

`--initdefaults` first creates the `/etc/mpss/default.conf` configuration file with default parameters, if such a file does not already exist.

Then, for each specified coprocessor, `micctrl --initdefaults` creates the `/etc/mpss/-micN.conf` configuration file with default parameters, if such a file does not already exist.
The micctrl utility

If such a default.conf or micN.conf file already exists, it is parsed for missing parameters, and default parameter values are added as needed. In addition, --initdefaults checks for deprecated parameters and replaces them with updated parameters. For example: the deprecated FileSystem parameter is updated to RootDevice RamFS. micctrl --initdefaults will not otherwise change an existing configuration if --users, --pass, --nocreate, --modhost, and --modcard suboptions are not specified.

--initdefaults then creates the common directory /var/mpss/common and creates and populates the per-coprocessor overlay hierarchy /var/mpss/micN for each specified coprocessor if these directories do not already exist.

If a /var/mpss/micN overlay hierarchy exists, it is parsed for missing files, and any missing files are added with values determined by the current configuration and micctrl --initdefaults suboptions.

For each user indicated by the --users suboption, and subject to the --nocreate suboption, --initdefaults copies ssh key files from the user’s host file system $HOME/.ssh to /var/mpss/micN/$HOME/.ssh. Next --initdefaults adds the user’s .pub keys (For Instance: from id_rsa.pub) to the /var/mpss/micN/$HOME/authorized_keys file.

B.4.2.2 Resetting configuration parameters

Command Syntax:

micctrl --resetdefaults [--vardir=<vardir>] [--srcdir=<srcdir>] \([-d|--netdir=<netdir>]
\([-u|--users=(none|overlay|merge|nochange)] \([-a|--pass=(none|shadow)] \([-c|--modhost=(yes|no)] \([-e|--modcard=(yes|no|<path_to_file>)]\]

Description:

The micctrl --resetdefaults command attempts to restore configuration parameters and the associated file systems to the default state. It shuts down the current network, removes a several files in the /etc directory of each specified coprocessor, removes the old configuration files and then calls the --initdefaults command. This process is intended to leave files created by the user untouched.

B.4.2.3 Cleaning configuration parameters

Command Syntax:

micctrl --cleanconfig [--vardir=<vardir>] [--srcdir=<srcdir>] \([-d|--netdir=<netdir>]

The micctrl --cleanconfig command is intended to completely remove all Intel® MPSS configuration artifacts.
B.4.2.3.1 Valid configuration file found

If a valid configuration is found, a series of steps are performed:

1. Shutdown the network and remove `micctrl` created network configuration files typically in `/etc/sysconfig/network-scripts` on RHEL* hosts and `/etc/sysconfig/networks` on SLES* hosts.

2. Remove all the files in the directory defined by the `MicDir` configuration parameter. Warning: this will also remove all the files in that directory not created by `micctrl`.

3. Remove the ramfs file or the NFS export directory associated with the NFS type in the `RootDevice` configuration parameter.

4. Remove the configuration file in `/etc/mpss` directory (or the directory defined by `$CONFIGDIR`).

5. If there are no more configuration files in the `/etc/mpss` directory, then remove the contents of the directory defined by the `CommonDir` parameter and remove the `default.conf` file from the `/etc/mpss` directory.

B.4.2.3.2 No valid configuration files

If no valid configuration file is found for a coprocessor, the following method of cleanup is performed:

1. Remove the entire contents of `/var/mpss/micN` for each specified coprocessor.

2. Delete file `/var/mpss/micN.image.gz`, for each specified coprocessor.

3. Delete file `/var/mpss/micN.export`, for each specified coprocessor.

4. Delete the `/etc/mpss/micN.conf` file for each specified coprocessor.

5. Delete the contents of the directory specified by the `CommonDir` parameter in the `/etc/mpss/default.conf` file.

6. Delete the `/etc/mpss/default.conf` file.

B.4.3 Setting the root device

The `micctrl --rootdev` command changes the configured `RootDevice` parameter which controls whether the coprocessor file root system will be mounted from a ram disk, or an NFS export.
B.4.3.1 RAM root file system

**Command Syntax:**

```bash
micctrl --rootdev=(RamFS|StaticRamFS) [--vardir=<vardir>] \ 
[(-t |--target=)<location>] [(-d|--delete)]
```

**Description:**

When the RootDev parameter type is either RamsFS or StaticRamFS, micctrl pushes a compressed CPIO archive to coprocessor memory at boot time, where it is uncompressed to become the coprocessor’s RAM file system.

For --rootdev=RamFS, micctrl sets the RootDevice parameter in the micN.conf of each specified coprocessor to:

```
RootDevice RamFS <location>
```

if --target is specified, or, otherwise, to

```
RootDevice RamFS /var/mpss/micN.image.gz
```

At boot time, micctrl builds a ram disk image from the files specified by the Base, CommonDir, Micdir, and Overlay configuration parameters. The resulting archive is saved as <ramfs_location>.

For --rootdev=StaticRamFS, micctrl sets the RootDevice parameter in the micN.conf of each specified coprocessor to:

```
RootDevice StaticRamFS <location>
```

if --target is specified, or, otherwise, to

```
RootDevice StaticRamFS /var/mpss/micN.image.gz
```

At boot time, there must be a previously created compressed CPIO archive at <ramfslocation> which will be used as the ram disk with which to boot the specified coprocessor(s).

If the current RootDevice parameter type is NFS or SplitNFS when micctrl --RamFS or micctrl --StaticRamFS is called with the --delete suboption, then the root and/or user file system hierarchies specified by the RootDevice configuration parameter are deleted.

B.4.3.2 NFS root file system

**Command Syntax:**

```bash
micctrl --rootdev=NFS [--vardir=<vardir>] \ 
[(-t |--target=)<host>:<location>] [(-c|--create)] \ 
[(-d|--delete)]
```

```bash
micctrl --rootdev=SplitNFS [--vardir=<vardir>] \ 
[(-t |--target=)<host>:<location>] [(-u|--usr=)<host>:<usr_location>] [(-c|--create)] \ 
[(-d|--delete)]
```
**Description:**

When the `RootDevice` parameter type is either `NFS` or `SplitNFS`, the file system of the specified coprocessors are remotely mounted from an NFS server(s).

For `--rootdev=NFS`, `micctrl` sets the `RootDevice` parameter in the `micN.conf` of each specified coprocessor to:

```
RootDevice NFS <share>
```

where `<share>` is set to:

```
<host>:<location>
```

if `--target=<host>:<location>`, or to:

```
<hostIP>:<location>
```

if `--target=<location>`, or to:

```
<hostIP>:/var/mpss/micN.export
```

if `--target` is not specified, and where `<hostIP>` is the IP address of the local host.

For `--rootdev=SplitNFS`, `micctrl` sets the `RootDevice` parameter in the `micN.conf` of each specified coprocessor to:

```
RootDevice SplitNFS <share> <usr_share>
```

where `<share>` is set is as for `--rootdev=NFS`, and where `<usr_share>` is set to:

```
<host>:<usr_location>
```

if `--usr=<host>:<location>`, or to:

```
<hostIP>:<usr_location>
```

if `--usr=<location>`, or to:

```
<hostIP>:/var/mpss/usr.export
```

if `--usr` is not specified, and where `<hostIP>` is the IP address of the local host.

It is the user’s responsibility to configure the specified or default location or locations for NFS export, typically in the specified host’s `/etc/exports` file. Generally each export specification should include `rw` and `no_root_squash` options.

If the `--create` suboption is specified, `micctrl` builds a root file system hierarchy from the files specified by the `Base`, `CommonDir`, `Micdir`, and `Overlay` configuration parameters and roots it at `<share>`. For `--rootdev=SplitNFS`, a file system hierarchy is also created and is rooted at `<usr_share>` it is a duplicate of `<share>/usr`. These hierarchies are only created if `<host>` is the local host. `micctrl` will not create these hierarchies on a remote host.

If the `--delete` suboption is specified, `micctrl` deletes the current root and user file system hierarchies.
**Note:** The `--server` suboption was previously used to enable specification of the `<server>` IP addressed. It has been deprecated and is only supported for backward compatibility.

### B.4.3.3 Rootdev configuration

**Command Syntax:**

```
micctrl --rootdev
```

**Description:**

When no type is specified, `micctrl --rootdev` outputs the current `RootDevice` configuration.

### B.4.3.4 Adding an NFS mount

**Command Syntax:**

```
micctrl --addnfs=[<host>]:<location> (-d |--dir=)<mount dir> \
[--options=<option>[,<option>]]
```

**Description:**

The `micctrl --addnfs` command adds an NFS mount entry, `<host>:<location>`, to the `/etc/fstab` file of each specified coprocessor. The optional `<host>`, if specified, must be a valid host name or host IP address. If `<host>` is not specified, it defaults to the local host.

The export will be mounted on the `<mount dir>` directory of each specified coprocessor. `micctrl` ensures that the mount directory is created on the coprocessor file system image.

The `--options` suboption specifies a list of NFS mount options. It must be a comma separated list in the standard form of the `/etc/fstab fs_mntops` field. Check NFS documentation for more information. The string supplied is placed into the options field in the coprocessors `/etc/fstab` file that `micctrl` creates for the added mount.

As with other NFS exports, it is the users responsibility to configure the specified `<location>` for NFS export.

**Additional configuration for SUSE* based host systems:**

If NFS file system mounts have been added and the `chkconfig` utility has been used to indicate starting the Intel® MPSS at host boot time, edit the `/etc/init.d/mpss` file and change the "# Required-Start:" line to read

```
"# Required-Start: nfsserver"
```

to ensure that NFS is started before the mpss service.

**Note:** A `--server` suboption was previously used to enable specification of the `<server>` IP addressed. It has been deprecated and is only supported for backward compatibility.
B.4.3.5 Removing an NFS mount

Command Syntax:

```
micctrl --remnfs=<mount dir>
```

Description:

The 'micctrl --remnfs' command searches the /etc/fstab files of the specified coprocessors for the entry corresponding to mount <mount dir>, and removes the mount point from the files.

B.4.3.6 Updating the compressed CPIO image

Command Syntax:

```
micctrl --updateramfs
```

Description:

When the RootDevice parameter of a specified coprocessor is RamFS or StaticRamFS, the micctrl --updateramfs command updates the coprocessor's current ram disk image with a new image built from the files specified by the base, commondir, micdir and overlay configuration parameters. The new image will be used the next time the coprocessor boots. The image file is saved at the location specified by the RootDevice parameter's <ramfs_location> value.

B.4.3.7 Updating NFS root exports

Command Syntax:

```
micctrl --updatenfs
micctrl --updateusr
```

Description:

When the RootDevice parameter of a specified coprocessor is NFS or SplitNFS, the micctrl --updatenfs command updates or builds a root file system hierarchy from the files specified by the Base, CommonDir, Micdir and Overlay configuration parameters and roots it at the location specified by the RootDevice parameters <share> value.

When the RootDevice parameter of a specified coprocessor is SplitNFS, the micctrl --updateusr command updates or builds a /usr file system hierarchy from the files specified by the Base, CommonDir, Micdir, and Overlay configuration parameters and roots it at the location specified by the RootDevice parameters <usr_share> value.
B.4.4 Configuring the coprocessor’s file system

B.4.4.1 Base file system location

Command Syntax:

```
micctrl --base
micctrl --base=default
micctrl --base=(cpio|dir) --new=<location>
```

Description:

The `micctrl --base` command modifies the `Base` parameter in the `/etc/mpss/micN.conf` configuration files of the specified coprocessors.

For `--base=cpio`, `micctrl` sets the `base` parameter to:

```
Base CPIO <location>
```

where `<location>` must be a compressed CPIO archive.

For `--base=dir`, `micctrl` sets the `Base` parameter to:

```
Base DIR <location>
```

where `<location>` is a ram file system hierarchy. If `<location>` does not exist, and the current `base` type is currently `CPIO`, then the corresponding CPIO image is expanded and files are extracted to `<location>`. If `<location>` does not exist, and the current `base` type is `DIR`, then the corresponding directory is copied to `<location>`.

For `--base=default`, `micctrl` resets the `Base` parameter to the default:

```
Base CPIO /usr/share/mpss/boot/initramfs-knightscorner.cpio.gz
```

For `--base` (For Example: a value is not specified), `micctrl` outputs the current `base`, `CommonDir` and `MicDir` parameter values.

B.4.4.2 Common files location

Command Syntax:

```
micctrl --commondir
micctrl --commondir=<commondir>
```

Description:

The `micctrl --commondir` command modifies the `CommonDir` configuration parameter for each specified coprocessor.

For `--commondir=<commondir>`, `micctrl` adds or modifies the `CommonDir` parameter in the `/etc/mpss/micN.conf` configuration file of each specified coprocessors. As a result, it overrides the `CommonDir` parameter in `/etc/mpss/default.conf`. The resulting parameter has the form:
CommonDir <commondir>

If the <commondir> directory does not exist, then it is created and the contents of the previous CommonDir directory are copied to the new location.

After the files have been copied, the configurations for all known coprocessors in the host are checked for references to the old CommonDir <commondir> directory. If no references exist, the files in that directory are deleted.

For --command (For Example: <commanddir> is not specified), micctrl outputs the current Base, CommonDir and MicDir parameter values.

Note: Previously, this command included a suboption to set a corresponding filelist associated with the files. The use of this file has been removed.

B.4.4.3 Coprocessor-specific files location

Command Syntax:

```
micctrl --micdir
micctrl --micdir=<micdir>
```

Description:

The micctrl --micdir command modifies the MicDir parameter in the /etc/mpss/micN.conf configuration file of each specified coprocessor.

For --micdir=<micdir>, micctrl modifies the MicDir parameter in the micN.conf configuration files of the specified coprocessors to:

```
MicDir <micdir>
```

If the <micdir> directory does not exist, it is created and the contents of the previous MicDir directory are copied to the new location. Finally, the previous MicDir directory is deleted.

For --micdir (For Example: <micdir> is not specified), micctrl outputs the current Base, CommonDir and MicDir parameter values.

Note: Previously, this command included a suboption to set a corresponding filelist associated with the files. The use of this file has been removed.

B.4.4.4 Additional file system overlays

Command Syntax:

```
micctrl --overlay
micctrl --overlay=(simple|file) (-s |--source=)<source> \ (-t |--target=)<target> (-d |--state=){on|off|delete)
micctrl --overlay=rpm (-s |--source=)<source> \ (-d |--state=){on|off|delete}
```
Description:

The `micctrl` --overlay command creates, modifies or deletes an Overlay parameter in the /etc/mpss/micN.conf configuration file of each specified coprocessor. The Overlay parameter describes a file or directory of files that are added to the coprocessors file system. There may be multiple Overlay parameters.

Note: Do not add overlays to the /tmp directory on the coprocessor, as it is cleared during each boot.

For --overlay=file and --state=on, `micctrl` appends a parameter:

    Overlay File <source> <target> on

to each /etc/mpss/micN.conf file. For --overlay=file and --state=off, `micctrl` appends a parameter:

    Overlay File <source> <target> off

to each /etc/mpss/micN.conf file. For --overlay=file and --state=delete, `micctrl` searches /etc/mpss/micN.conf for the parameter:

    Overlay File <source> <target> (on|off)

and removes it if found.

For --overlay=simple and --state=on, `micctrl` appends a parameter:

    Overlay Simple <source> <target> on

to each /etc/mpss/micN.conf file. For --overlay=simple and --state=off, `micctrl` appends a parameter:

    Overlay Simple <source> <target> off

to each /etc/mpss/micN.conf file. For --overlay=simple and --state=delete, `micctrl` searches /etc/mpss/micN.conf for the parameter:

    Overlay Simple <source> <target> (on|off)

and removes it if found.

For --overlay=rpm and --state=on, `micctrl` appends a parameter:

    Overlay RPM <source> on

to each /etc/mpss/micN.conf file. For --overlay=rpm and --state=off, `micctrl` appends a parameter:

    Overlay RPM <source> off

to each /etc/mpss/micN.conf file. For --overlay=rpm and --state=delete, `micctrl` searches /etc/mpss/micN.conf for the parameter:

    Overlay RPM <source> (on|off)
and removes it if found.

For --overlay (no overlay type specified), micctrl outputs the currently defined overlays.

You can also add Overlay parameters to a user created configuration file by directly editing the file. The Include configuration parameter can be used to include such a file. micctrl does not modify such user created configuration files. To override an Overlay parameter in such a configuration file without editing the file, you can call micctrl --overlay to add an Overlay parameter to micN.conf that changes the state of a specified overlay to off or on as needed.

**Note:** The filelist overlay type has been deprecated and is only supported for backward compatibility; only files owned by root are supported. Use the simple and file overlay types instead.

**Note:** The state=off suboption has been deprecated and is only supported for backward compatibility.

### B.4.4.5 Location of additional RPMs for the coprocessor

**Command Syntax:**

```
micctrl --rpmdir=<location>
```

**Description:**

The `micctrl --rpmdir` command sets the K1omRpms configuration parameter in the micN.conf configuration file of the specified coprocessors to the specified <location>. See Appendix A.4.3 for information on the K1omRpms parameter.

### B.4.5 Networking configuration

Several `micctrl` commands aid in configuring coprocessor networking.

**Note:** On SUSE* hosts, run `service networking restart` upon completion of all network change commands.

#### B.4.5.1 MAC address assignment

**Command Syntax:**

```
micctrl --mac=(serial|random|<MAC address>) \ 
[(-d|--netdir=<netdir>) [(-w|--distrib)=(redhat|suse)]
```

**Description:**

The `micctrl --mac` command modifies the MacAddrs configuration parameter in the micN.conf configuration file of each specified coprocessor. The MacAddrs parameter defines the method for setting the MAC addresses of both the host and the coprocessor endpoints.
For --mac=serial, *micctrl* sets *MacAddrs* to:

```
MacAddrs Serial
```

For --mac=random, *micctrl* sets *MacAddrs* to:

```
MacAddrs Random
```

For --mac=<MAC address>, and where <MAC address> is any valid MAC address in the format XX:XX:XX:XX:XX:XX, and X is an ASCII hex digit (0..F), *micctrl* sets the *MacAddrs* parameter of the first specified coprocessor to:

```
```

the *MadAddrs* parameter of the second specified coprocessor to:

```
```

the *MadAddrs* parameter of the Nth specified coprocessor to:

```
MacAddrs XX:XX:XX:XX:XX:(XX+2*(N-1)+1) XX:XX:XX:XX:XX:(XX+2*(N-1))
```

For example, if the least significant octet of <MAC address> is '08', then *micctrl* sets the *MacAddrs* parameter of the first specified coprocessor to:

```
```

the *MadAddrs* parameter of the second specified coprocessor to:

```
MacAddrs XX:XX:XX:XX:XX:0B XX:XX:XX:XX:XX:0A
```

the *MadAddrs* parameter of the Nth specified coprocessor to:

```
MacAddrs XX:XX:XX:XX:XX:(08+2*(N-1)+1) XX:XX:XX:XX:XX:(08+2*(N-1))
```

### B.4.5.2 Resetting the network to a default configuration

**Command Syntax:**

```
micctrl --network=default
```

**Description:**

The *micctrl* --network=default command restores the network configuration for the specified coprocessors to the default (Static Pair).

### B.4.5.3 Static pair

**Command Syntax:**

```
micctrl --network=static [(-d|--netdir)<netdir>] \\
[(-w|--distrib)=(redhat|suse)] [(-i|--ip)<ip>] \\
```

The micctrl utility

[-n|--netbits=<netbits>] [-m|--mtu=<mtu>] \
[-c|--modhost=](yes|no) [-e|--modcard=](yes|no|<path_to_file>)

Description:
The static pair network topology is configured using the **micctrl --network** command. This topology is described in [Section 2.2.3.1](#).

The **micctrl --network** command modifies the **Network** parameter of each specified coprocessor. This command also creates and/or modifies host and coprocessor network configuration files, and brings network endpoints on the host down and up as needed. That process is described in detail in [Section 5.2.1.2](#).

When the **--bridge** suboption is not specified, the **micctrl --network=static** command configures the static pair network topology between the host and each specified coprocessor.

There are several alternatives for setting IP addresses. If the **--ip** suboption is not given, then IP addresses are as assigned by **micctrl --initdefaults**. Refer to [Appendix A.5.3](#) for details.

If the **--ip** suboption is given and **<ip>** is two quads (XX.XX), then **micctrl** uses those as the high order quads of IP addresses which it constructs. The third quad of each such address is N + 1 for each coprocessor with a name specified to micN. The fourth quad of each coprocessor endpoint address is 1, and the fourth quad of each host endpoint address is 254. For example, on a two coprocessor system, the suboption **--ip=172.31** will result in addresses 172.31.1.1 and 172.31.1.254 for mic0’s coprocessor and host endpoints, and 172.31.2.1 and 172.31.2.254 for mic1’s coprocessor and host endpoints.

Fully qualified IP addresses can be assigned. In this case, **<ip>** must have the format **cardIP,hostIP:cardIP,hostIP:** and so on. Each **cardIP,hostIP** pair specifies the IP address for one static pair network, where the first pair is the IP address of the network between the host and the first specified coprocessor. For example, if there are two cards in the system, the suboption **--ip=172.31.10.1,172.31.10.2:172.3.11.1,172.31.11.2** results in the first specified coprocessor and host endpoints having addresses **172.31.10.1** and **172.31.10.2** and the second specified coprocessor and host endpoints having addresses **172.31.11.1** and **172.31.11.2**.

The **--mtu** suboption sets the virtual network packet size to **<mtu>** bytes. The default mtu size of mtu is 64KB. Testing has shown that the default value yields the best performance for this network type.

The **--netbits** suboption defines a netmask. If fully qualified IP addresses are assigned, the addresses must be identical over the high order **<netbits>** bits. The default value is 24, defining a netmask of 255.255.255.0. There is rarely any need to change this parameter.
B.4.5.4 Internal bridging

Command Syntax:

```
micctrl --addbridge=<brname> --type=internal \ 
  (-i | --ip=)<bridge_ip> [{(-d | --netdir=)<netdir>} \ 
  [(-w | --distrib={Redhat|suse}) \ 
  [(-n | --netbits=)<netbits>] \ 
  [(-m | --mtu=)<mtu>] ]

micctrl --network=static --bridge=<name> --ip=<mic_ip> \ 
  [(-c | --modhost=){yes|no}] [(-e | --modcard=) \ 
  {yes|no|<path_to_file>}]```

Description:

The internal bridge network topology is configured using the `micctrl --addbridge` and `--network` commands. This topology is described in Section 2.2.3.2.1. The bridge interface is created first, and is then connected to the virtual network interfaces of each specified coprocessor.

--addbridge suboptions:

The `micctrl --addbridge` and `--network` commands modify the Bridge parameter common to all specified coprocessors, and the Network parameter of each specified coprocessor. These commands also create and/or modify host and coprocessor network configuration files, and bring network endpoints on the host down and up as needed. That process is described in detail in Section 5.2.2.2.

The `micctrl --addbridge` command creates the bridge interface. The bridge name, `<brname>`, of the bridge must be specified. The `--type=internal` suboption causes `micctrl` to create the correct network configuration files for an internal bridge.

The bridge IP address, `<bridge_ip>`, must be a fully qualified dot notated address.

The `--mtu` suboption sets the virtual network packet size to `<mtu>` bytes. The default mtu size of mtu is 64KB. Testing has shown that the default value yields the best performance for this network type.

The `--netbits` suboption defines a netmask. The bridge IP address and all coprocessor endpoint IP addresses must be identical over the high order `<netbits>` bits. The default value is 24, defining a netmask of 255.255.255.0. There is rarely any need to change this parameter.

`micctrl --addbridge` creates the bridge configuration file, for example `$NETDIR/ifcfg-br0`, if it does not already exist. If the bridge configuration file already exists, then `<bridge_ip>`, `<netbits>`, and `<mtu>` must match the corresponding values of the specified bridge.

--network suboptions:

The `micctrl --network` command adds coprocessor virtual network interfaces to the bridge. The `--bridge=<name>` argument is required, and the `<name>` must be the same as the `<brname>`, the name specified to `--addbridge`.

The bridge’s mtu and netbits values are used in configuring coprocessor virtual network interfaces.
B.4.5.5 External bridging

The external bridge network topology is configured using the `micctrl --addbridge` and `--network` commands. This topology is described in Section 2.2.3.2. The bridge interface is created first, and is then connected to the virtual network interfaces of each specified coprocessor.

The `micctrl --addbridge` and `--network` commands modify the Bridge parameter common to all specified coprocessors, and the Network parameter of each specified coprocessor. These commands also creates and/or modifies host and coprocessor network configuration files, and brings network endpoints on the host down and up as needed. That process is described in detail in Section 5.2.3.2.

Because external bridging gives coprocessors access to the external network, DHCP based IP address assignment is supported for this topology.

B.4.5.5.1 External bridging, static ip address assignment

**Command Syntax:**

```
micctrl --addbridge=<brname> --type=external \ 
(-i|--ip)=<bridge_ip> [(-d|--netdir)=<netdir>] \ 
[(-w|--distrib)=(redhat|suse)] \ 
[(-n|--netbits)=<netbits>] \ 
[(-m|--mtu)=<mtu>]

micctrl --network=static --bridge=<name> --ip=<mic_ip> \ 
[(-c|--modhost)=(yes|no)] [(-e|--modcard)=(yes|no|<path_to_file>)]
```

**Description:**

**--addbridge suboptions:**

For the static IP address assignment case, `micctrl --addbridge` and `micctrl --network` commands are the same as for internal bridging with the exception that the bridge type is `external`.

The `--type=external` suboption causes `micctrl` to create the correct network configuration files for an external bridge.

The bridge IP address, `<bridge_ip>`, must be a fully qualified dot notated address.

The `--mtu` suboption sets the virtual network packet size to `<mtu>` bytes. The default mtu size of mtu is 1500B for compatibility with typical external networks.

The `--netbits` suboption defines a netmask. The bridge IP address and all coprocessor endpoint IP addresses must be identical over the high order `<netbits>` bits. The default value is 24, defining a netmask of 255.255.255.0. There is rarely any need to change this parameter.

`micctrl --addbridge` creates the bridge configuration file, for example `$NETDIR/ifcfg-br0`, if it does not already exist. If the bridge configuration file already exists, then `<bridge_ip>`, `<netbits>`, and `<mtu>` must match the corresponding values of the specified bridge.
The micctrl utility

```
--network suboptions:
```

The `micctrl --network` command adds coprocessor virtual network interfaces to the bridge. The `--bridge=<name>` argument is required, and the `<name>` must be the same as the `<brname>`, the name specified to `--addbridge`. The `--ip` argument to `--network` is also required, and `<mic_ip>` must be a fully qualified dot notated IP address in which the first 3 quads match those of the bridge IP address, `<bridge_ip>`. If more than one coprocessor is specified, each will be assigned the specified `<mic_ip>` with the coprocessor’s number added to the fourth quad. For example, for `--ip=172.31.10.12`, mic0 will be assigned the address 172.31.10.12 and mic1 will be assigned the address 172.31.10.13.

The bridge’s `mtu` and `netbits` values are used in configuring coprocessor virtual network interfaces.

It is the user’s responsibility to slave the physical Ethernet endpoint to the bridge. For example, on RHEL*, the line "BRIDGE=br0" is added to the `eth0` Ethernet configuration file, `/etc/sysconfig/network-scripts/ifcfg-eth0` to connect endpoint `eth0` to bridge `br0`:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

On SLES* host platforms, the physical port name must be added to the `BRIDGE_PORTS` entry in the `/etc/sysconfig/networks/ifcfg-br0` configuration file, for example:

```
BRIDGE_PORTS='eth0 mic0 mic1'
```

**B.4.5.5.2 External bridging, dhcp address assignment**

**Command Syntax:**

```
micctrl --addbridge=<brname> --type=external --ip=dhcp \[(--d |--netdir=<netdir>)\] \[(--w |--distrib=)(redhat|suse)\] 
micctrl --network=dhcp --bridge=<name> \[(--c |--modhost=)(yes|no)\] \[(--e |--modcard=)\] \[(yes|no<path_to_file>)\]
```

**Description:**

`--addbridge suboptions:`

DCHP address assignment is configured by setting the `micctrl --addbridge` command’s `--ip` value and the `--network` type to `dhcp`. During coprocessor boot, the coprocessor Linux* OS will attempt to retrieve an IP address from a DHCP server. The DHCP server will also configure netbits and mtu values.

`micctrl --addbridge` creates the bridge configuration file, for example `$NETDIR/ifcfg-br0`, if it does not already exist.
It is the user’s responsibility to slave the physical Ethernet endpoint to the bridge. For example, on RHEL*, the line "BRIDGE=br0" is added to the eth0 Ethernet configuration file, /etc/sysconfig/network-scripts/ifcfg-eth0 to connect endpoint eth0 to bridge br0:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

On SLES* host platforms, the physical port name must be added to the BRIDGE_PORTS entry in the /etc/sysconfig/networks/ifcfg-br0 configuration file, for example:
```
BRIDGE_PORTS='eth0 mic0 mic1'
```

The modhost and modcard parameters are not needed for configuring host and coprocessor /etc/hosts files in the case that a name server is available from which coprocessor and host IP addresses can be retrieved.

### B.4.5.6 Changing network parameters

**Command Syntax:**
```
micctrl --network [{(-d|--netdir)=<netdir>]} \ 
[(-w|--distrib)=(redhat|suse)] [{(-i|--ip)=<ip>}] \ 
[(-n|--netbits)=<netbits>] [{(-m|--mtu)=<mtu>}] \ 
[(-c|--modhost)=(yes|no)] [{(-e|--modcard)=} \ 
(yes|no|<path_to_file>)]
```

**Description:**

The `micctrl --network` command (with no network type specified) may be used to change the parameters for a set of interfaces.

### B.4.5.7 Modifying a bridge

**Command Syntax:**
```
micctrl --modbridge=<brname> [{(-d|--netdir)=<netdir>}] \ 
[(-w|--distrib)=(redhat|suse)] [{(-i|--ip)=<ip>}] \ 
[(-n|--netbits)=<bits>] [{(-m|--mtu)=<mtu>}] 
```

**Description:**

The `micctrl --modbridge` command modifies the IP address, netbits and/or MTU values of the specified network bridge. In addition any changed netbits or MTU values are propagated to any of the attached virtual network configuration files.

The --ip suboption sets the bridge’s IP address. `<bridge_ip>` must be a fully qualified dot notated address.

The --mtu suboption sets the virtual network packet size to `<mtu>` bytes. The default mtu size of mtu is 64KB. Testing has shown that the default value yields the best performance for this network type.
The --netbits suboption defines a netmask. The bridge IP address and all coprocessor endpoint IP addresses must be identical over the high order <netbits> bits. The default value is 24, defining a netmask of 255.255.255.0. There is rarely any need to change this parameter.

**B.4.5.8 Deleting a bridge**

**Command Syntax:**

```
micctrl --delbridge=<brname> [(-d|--netdir=<netdir>) \ 
[(-w|--distrib=(redhat|suse)])
```

**Description:**

The `micctrl --delbridge` command removes a specified bridge from the coprocessor configuration. If the specified bridge is marked as internal, the corresponding host network configuration file will be deleted.

All coprocessors must have been detached from the bridge before the bridge can be deleted. The `micctrl --network=default` command can be used for this purpose.

**B.4.6 User credentialing on the coprocessor**

**B.4.6.1 Updating user credentials**

**Command Syntax:**

```
micctrl --userupdate=(none|overlay|merge|nochange) \ 
[(-a|--pass=(none|shadow)) [--nocreate]
```

**Description:**

The `micctrl --userupdate` command enables updating certain user credential information.

For `--userupdate=none`, the `/etc/passwd` and `/etc/shadow` files are recreated with the minimal set of users required by Linux*, which are the root, ssh, nobody, nfsnobody and micuser.

For `--userupdate=overlay`, the `/etc/passwd` and `/etc/shadow` files are recreated with the users from the `--userupdate=none` suboption and any regular users found in the `/etc/passwd` file of the host.

For `--userupdate=nochange`, behavior is as for `--userupdate=overlay` if no configuration exists for the specified coprocessor. Otherwise the `/etc/passwd` and `/etc/shadow` files are unchanged.

For `--userupdate=merge`, any users in the host’s `/etc/passwd` file but not in the specified coprocessor’s `/etc/passwd` file are added to the coprocessor’s `/etc/passwd` and `/etc/shadow` files.

**B.4.6.2 Adding users**

**Command Syntax:**
The micctrl utility

micctrl --useradd=<user> [(-u|--uid=<uid>] \ 
[(-g|--gid=<gid>] [(-d|--home=<dir>] \ 
[(-c|--comment=<string>] [--shell=<shell>] \ 
[(-k|--sshkeys=<keydir>] [--nocreate] [--non-unique]

**Description:**

The `micctrl --useradd` command adds the user named `<user>` to the `/etc/passwd` and `/etc/shadow` files in the directory identified by the `MicDir` parameter of each specified coprocessor.

The `--uid` suboption specifies the user ID of user `<user>`. By default, the user ID of user `<user>` on the host is used.

The `--gid` suboption specifies the group ID of user `<user>`. By default, the group ID of user `<user>` on the host is used.

The `--home` suboption specifies the home directory in the coprocessor file system of user `<user>`. By default, the home directory is `/home/<user>`.

The `--comment` suboption specifies a comment string to be added to the comment field of the `/etc/passwd` entry for user `<user>`. The default comment string is `<user>`.

The `--shell` suboption replaces the default shell used to login. If it is not specified and user exists on the host system shell will be copied from the host. Otherwise it will set The default to `/bin/bash` shell.

The `--sshkeys` suboption specifies the host directory in which the user's secure shell key files are to be found. The default is `/home/<user>/.ssh`. The '*.pub' public SSH keys are copied to the .ssh directory in the user’s home directory of the coprocessor file system.

The `--non-unique` suboption will allow the user to be added to the coprocessor’s /etc/passwd and /etc/shadow files with the specified uid even if a user with that uid already exists.

A default .profile file is created in the user’s home directory of the coprocessor file system home directory.

The user is also added to the /etc/passwd and /etc/shadow files of each specified coprocessor that is in the online state. In addition, a home directory is created if the --nocreate suboption is not specified, and the user’s SSH keys are pushed to the user’s home directory.

---

**B.4.6.3 Removing users**

**Command Syntax:**

```
micctrl --userdel=<user> [(-r|--remove)]
```

**Description:**
The `micctrl --userdel` command removes the user named `<user>` from the `/etc/passwd` and `/etc/shadow` files in the directory identified by the `MicDir` parameter of each specified coprocessor.

By default, `--userdel` does not remove the user’s home directory on the coprocessor; this is intended to prevent the inadvertent removal of a user’s remote mounted home directory. Home directory removal can be forced by including the `--remove` suboption.

### B.4.6.4 Changing user’s password

**Command Syntax:**

```
micctrl --passwd

micctrl --passwd=<user> [(-s |--stdin)]
```

**Description:**

The `micctrl --passwd` command changes a user’s password in the `/etc/shadow` file in the directory identified by the `MicDir` parameter of each specified coprocessor.

A non-superuser calls `micctrl --passwd` with no name, and is prompted for the current password and then for the new password.

The superuser specifies a user’s name, `<user>`, when calling `micctrl --passwd`. If the `--stdin` suboption is not specified then `micctrl` will be prompted to provide a new password for a user and to confirm it.

The `-p --pass` option was deprecated, superuser may use the `--stdin` suboption to pass a new password in the standard input:

```
micctrl --passwd=username --stdin <<< password
```

or using pipe from a different program:

```
echo “newPassword” | micctrl --passwd=username --stdin
```

`echo` should be a shell built-in. If it resolves to `/bin/echo` the password may be visible to other users.

The `--stdin` option is considered as less secure and should be used only in justified cases. It is also advised to pass the password using pipe from different program.

The user account on each specified coprocessor that is in the online state will be updated with the password.

### B.4.6.5 Adding groups

**Command Syntax:**

```
micctrl --groupadd=<name> (-g |--gid=<gid>)
```

**Description:**
The micctrl utility

The *micctrl* --groupadd command adds the specified group name and ID to the */etc/group* file in the directory identified by the *MicDir* parameter of each specified coprocessor.

The group will also be added to the */etc/group* file of each specified coprocessor that is in the *online* state.

**B.4.6.6 Removing groups**

**Command Syntax:**

```bash
micctrl --groupdel=<name>
```

The *micctrl* --groupdel command removes the specified group name, along with its ID, from the */etc/group* file in the directory identified by the *MicDir* parameter of each specified coprocessor.

The group will also be deleted from the */etc/group* file of each specified coprocessor that is in the *online* state.

**B.4.6.7 Specifying the host secure shell keys**

**Command Syntax:**

```bash
micctrl --hostkeys=<keydir>
```

**Description:**

The *micctrl* --hostkeys command copies files from the *<keydir>* directory to the *$MICDIR/etc/ssh* file of each specified coprocessor.

*micctrl* --initdefaults generates a set of ssh key files in the */etc/ssh* directory of each specified coprocessor, in the directory identified by the *MicDir* parameter. The keys in this directory identify a coprocessor as a “known host” during ssh operations if there is a match to the user’s *known_hosts* file (typically in *$HOME/.ssh*).

If a configuration is completely regenerated, such as by calling *micctrl* --cleanconfig followed by *micctrl* --initdefaults, the user’s *known_hosts* will have to be revised to match the new set(s) of host keys. To avoid the need to do this, the existing sets of host keys can be saved before regenerating the configuration to some *<keydir>* directory, and then restored afterward using the *micctrl* --hostkeys command.

**B.4.6.8 Updating user’s ssh keys**

**Command Syntax:**

```bash
micctrl --sshkeys [-d|--dir=<dir>] 
micctrl --sshkeys=<user> [-d|--dir=<dir>]
```

**Description:**
The `micctrl --sshkeys` command copies a set of *.pub public ssh keys to the $HOME/.ssh directory of some user in the file system of each specified coprocessor.

A non-root user does not specify a `<user>` to `micctrl --sshkeys`. All *.pub public keys are copied to that user's $HOME/.ssh directory. Key files are copied from `<dir>` if specified, otherwise from the user's $HOME/.ssh directory on the host. Only files owned by the user are copied.

A root-user specifies a `<user>` to `micctrl --sshkeys`. Any *.pub Keys are copied to that user's $HOME/.ssh directory. Key files are copied from `<dir>` if specified, otherwise from the user's $HOME/.ssh directory on the host. Only files owned by the specified user are copied.

`micctrl --sshkeys` will also use add any *.pub files to the 'authorized_keys' file if not already present.

**B.4.6.9 Configuring LDAP**

**Command Syntax:**

```
micctrl --ldap=(<server>|disable) (-b |--base=)<domain>
```

**Description:**

The `micctrl --ldap` command configures the coprocessor to use LDAP for user authentication.

For `--ldap=<server>`, `micctrl` configures LDAP to use the `<server>` as the authentication server and configures `<domain>` as the domain.

For `--ldap=disable`, `micctrl` disables LDAP service on each specified coprocessor.

When this command is called, the `K1omRpms` configuration parameter must be set as needed.

**B.4.6.10 Configuring NIS**

**Command Syntax:**

```
micctrl --nis=(<server>|disable) (-d |--domain=)<domain>
```

**Description:**

The `micctrl --nis` command configures the coprocessor to use NIS for user authentication.

For `--nis=<server>`, `micctrl` configures NIS to use the `<server>` as the NIS/YP server and configures `<domain>` as the domain.

For `--nis=disable`, `micctrl` disables NIS service on each specified coprocessor.

When this command is called, the `K1omRpms` configuration parameter must be set as needed.
B.4.7 Configuring the coprocessor OS kernel

B.4.7.1 The coprocessor OS image location

Command Syntax:

```
micctrl --osimage
micctrl --osimage=<osimage> (-s|--sysmap=)<sysmapfile>
```

Description:

The `micctrl --osimage` command sets the OSimage parameter in the `micN.conf` configuration file of each specified coprocessor to `<osimage> <sysmapfile>`. The `<osimage>` argument is the Linux* operating system image to be booted, and `<sysmapfile>` identifies the matching system map file which holds values used by the mpssd daemon.

For `--osimage` (For Example: no `<osimage>` value is specified), `micctrl` outputs the current OSimage parameter value for each specified coprocessor.

B.4.7.2 Booting on the Intel® MPSS service start

Command Syntax:

```
micctrl --autoboot
micctrl --autoboot=(yes|no)
```

Description:

The `micctrl --autoboot` command sets the BootOnStart configuration parameter to the specified value.

For `--autoboot` (For Example: no `--autoboot` value is specified), `micctrl` outputs the current BootOnStart value for each specified coprocessor.
B.4.7.3 Power management configuration

Command Syntax:

```
micctrl --pm
micctrl --pm=(set|default|defaultb|off) \ 
[(-c |--corec6)=(on|off)] \ 
[(-t |--pc3)=(on|off)] \ 
[(-s |--pc6)=(on|off)] \ 
[(-f |--cpufreq)=(on|off)]
```

Description:

The `micctrl --pm` command sets the `PowerManagement` configuration parameter for each specified coprocessor.

For `--pm=set`, the power management parameters are set as specified by the optional arguments `--corec6`, `--pc3`, `--pc6`, and `--cpufreq`. Each optional parameter can be individually enabled or disabled by setting the `on` or `off` values.

For `--pm=default`, the power management configuration is set to the default for each coprocessor for which the stepping can be determined. If the stepping of a coprocessor cannot be determined, its power management configuration is set to the default for C stepping.

For `--pm=defaultb`, the power management configuration for each specified coprocessor is set to the default for B stepping.

For `--pm=off`, all parameters other than `cpufreq` are set to the off state.

For `--pm` (For Example: no `--pm` value is specified), `micctrl` outputs the current `PowerManagement` value for each specified coprocessor.

Note: It is recommended to use the default power management settings unless directed by an Intel® representative to change them.

B.4.7.4 Cgroups configuration

Command Syntax:

```
micctrl --cgroup [(-m |--memory)=(enable|disable)]
```

Description:

The `micctrl --cgroup` command modifies the `Cgroup` parameter for each specified coprocessor to value of the `--memory` suboption.

If the `--memory` suboption is not specified, `micctrl` outputs the current value of the `Cgroup` parameter of each specified coprocessor.
B.4.7.5 Syslog configuration

**Command Syntax:**

```shell
micctrl --syslog
micctrl --syslog=buffer [(-l |--loglevel=)<loglevel>]
micctrl --syslog=file [(-f |--logfile=)<location>] \
[(-l |--loglevel=)<loglevel>]
micctrl --syslog=remote (-s |--host=)<targethost[:port]> \
[(-l |--loglevel=)<loglevel>]
```

**Description:**

The `micctrl --syslog` command creates and/or modifies the `/etc/syslog-startup.conf` file in the filesystem of each specified coprocessor.

For `--syslog=buffer`, syslog is only available from the kmsg buffer.

For `--syslog=file`, the syslog daemon logs to the optional `<location>` or to the `/var/log/messages` log file.

For `--syslog=remote`, the syslog daemon is instructed to log to the remote node specified by the optional `host` argument. The `port` value defaults to 514. If the `--host` suboption is not specified then the remote host defaults to `host:514`.

For `--syslog` (no `--syslog` type is specified), the current syslog configuration is output.

Changes to the logfile location take effect immediately on each specified coprocessor that is in the `online` state.

**Note:** `micctrl --syslog` only configures syslog on the coprocessor. Remote host may need additional configuration. Please refer to the documentation of your host logger daemon to determine how to enable collecting logs from remote hosts.

B.4.8 Deprecated micctrl commands

B.4.8.1 --service command

**Command Syntax:**

```shell
micctrl --service
micctrl --service=<name> --state=(on|off) [--start=<num>] \ 
[--stop=<num>] [mic card list]
```

**Description:**

The coprocessor OS, like any Linux* OS, executes a series of scripts on boot, which are located in `/etc/init.d`. To determine which of the installed scripts are executed on any boot, links to these scripts are created in runlevel directory. The coprocessor’s OS runs at level 5, defining the runlevel directory to be `/etc/rc5.d`.

On most Linux* systems, the service scripts to be executed are enabled or disabled using the `chkconfig` command. On the Intel® MPSS this is performed by the `micctrl --service` command.
The --state suboption must be set to on or off and determines whether the script will execute on boot. Services already included in the configuration may have their state changed without specifying new start or stop values.

The start and stop parameters must be between 1 and 100, and determine the order in which the services are executed. If stop is not specified, then it will be set to 100 – start.

Add on software containing a service script will include the Service parameter associated with it. Modifying the default value included in its own configuration file will cause an overriding entry to be set in the micN.conf file.

micctrl --service may be called with no arguments and will display a list of current service settings. Currently, no services are configured by default.

B.4.8.2 --configuser command

Command Syntax:

```
micctrl --configuser=none [-ids] [mic card list]
micctrl --configuser=local [--low=<low uid>] [--high=<high uid>]
\ [-ids] [mic card list]
```

Description:

This command has been removed. Refer to the section on the micctrl --userupdate command for its functional replacement.

B.4.8.3 --resetconfig command

Command Syntax:

```
micctrl --resetconfig [--users=(none|overlay|merge|nochange) \ [--pass=(none|shadow)] [--nocreate] [--modhost=(yes|no)] \ [--modcard=(yes|no|<path_to_file>)] [mic card list]
```

Description:

Changes to the configuration files are propagated with the micctrl --resetconfig command. The --resetconfig command first removes the files in MicDir created by the configuration process, with the exception of the highly persistent ssh host key files. It then regenerates those files according to the parameters in the /etc/mpss/micN.conf and /etc/mpss/default.conf files. This process will not add default parameters, but only causes the changed parameters to be propagated. The --resetconfig command added several new options with the 3.2 release. Consult the previous documentation for the --initdefaults command.
C Intel® MPSS host driver sysfs entries

The mic.ko driver supplies configuration and control information to host software through the Linux* Sysfs file system. The driver presents two sets of information:

- Driver global information is presented in the /sys/class/mic/ctrl directory.
- Information unique to a coprocessor instance is presented in the /sys/class/mic/micN directories.

C.1 The global mic.ko driver sysfs entries

C.1.1 Revision information

Sysfs Entries:

/sys/class/mic/ctrl/version

This entry is read-only. The version sysfs entry displays a string containing the ID of the build producing the current installed software.

C.1.2 Other global entries

Sysfs Entries:

/sys/class/mic/ctrl/peer2peer
/sys/class/mic/ctrl/vnet

These entries are read-only.

The peer2peer sysfs entry reports the state – enable or disable, of Symmetric Communication Interface (SCIF) based communication between coprocessors, referred to as peer-to-peer (p2p) communication.

On reading, the vnet entry returns the number of active links to the virtual Ethernet.

C.2 The mic.ko driver sysfs entries

C.2.1 Hardware information

Sysfs Entries:

/sys/class/mic/micN/family
/sys/class/mic/micN/sku
/sys/class/mic/micN/stepping
/sys/class/mic/micN/active_cores
/sys/class/mic/micN/memsize

These sysfs entries are all read-only.

The family node reports the coprocessor family. At this time the family should always report the string x100.

The sku node returns a string defining the device type, for example: C0-3120/3120A.

The stepping node returns the processor stepping, for example: B0, B1, or C0.

The active_cores node reports (base 16) the number of working cores on the coprocessor.

The memsize node returns the size of memory (in hexadecimal) on the coprocessor.

C.2.2 State entries

Sysfs Entries:
/sys/class/mic/micN/state
/sys/class/mic/micN/mode
/sys/class/mic/micN/image
/sys/class/mic/micN/cmdline
/sys/class/mic/micN/kernel_cmdline

The state and cmdline nodes are read/write. The others are read-only.

On reading, the state node reports one of the following values:
- ready: coprocessor is ready for a boot command
- booting: coprocessor is currently booting
- no response: coprocessor is not responding
- boot failed: coprocessor failed to boot
- online: coprocessor is currently booted
- shutdown: coprocessor is currently shutting down
- lost: booted coprocessor is not responding
- resetting: coprocessor is processing soft reset
- reset failed: coprocessor cannot be reset – non recoverable

Additionally, if the state is booting, online or shutdown, the state is modified by the information from the mode and image sysfs nodes. The mode will be either linux or elf. The image file will report the name of the file used to boot into the associated mode.
Writing to the *state* node requests the driver to initiate a change in state. The allowable requests are to boot, reset or shutdown the coprocessor.

To boot a coprocessor, the string to write has the format "*boot:*linux:*<image name>*". The *mpsd* daemon uses its *OSimage* parameter to fill in the image name. For example the default Linux* image for the coprocessor will create the string "*boot:*linux:*/usr/share/mpss/boot/bzImage-2.6.38.8". After a successful boot the *state* will indicate *online*, *mode* linux, and *image* /usr/share/mpss/boot/bzImage-2.6.38.8.

The *cmdline* parameter is set by user software, normally the *mpsd* daemon or *micctrl* utility, to pass kernel command line parameters to the coprocessor Linux* boot process. Current parameters include root file system, console device information, power management options and verbose parameters. When the *state* sysfs node requests the coprocessor to boot, the driver adds other kernel command line information to the string and records the complete string that was passed to the booting embedded Linux* OS in the *kernel_cmdline* sysfs node.

### C.2.3 Statistics

Sysfs Entries:

/sys/class/mic/micN/boot_count

/sys/class/mic/micN/crash_count

These entries are read-only. The *boot_count* sysfs node returns the number of times the coprocessor has booted to the online state. The *crash_count* sysfs node records the number of times that the coprocessor has crashed.

### C.2.4 Debug entries

Sysfs Entries:

/sys/class/mic/micN/platform

/sys/class/mic/micN/post_code

/sys/class/mic/micN/scif_status

/sys/class/mic/micN/log_buf_addr

/sys/class/mic/micN/log_buf_len

/sys/class/mic/micN/virtblk_file

The *platform*, *post_code* and *scif_status* entries are read-only; the *log_buf_addr*, *log_buf_len*, and *virtblk_file* entries are read and write.

The *platform* sysfs node should always return a zero value.

The *post_code* sysfs node returns the contents of the hardware register containing the state of the boot loader code. Reading it always returns two ASCII characters. Possible values of note are the strings "12", "FF" and any starting with the character '3'. A string of "12" indicates the coprocessor is in the ready state and waiting for a command to start executing. A string of "FF" indicates the coprocessor is executing code. A string
starting with the character ‘3’ indicates the coprocessor is in the process of training memory. Any other value should be transitory. Any other value remaining for any length of time indicates an error and should be reported to Intel®.

The `log_buf_addr` and `log_buf_len` parameters inform the host driver of the memory address in the coprocessor memory at which to read its Linux* kernel log buffer. The correct values to set are found by looking for the strings "log_buf_addr" and "log_buf_len" in the Linux* system map file associated with the file in the OSimage parameter, and are typically set by the `mpssd` daemon.

The `virtblk_file` sysfs node indicates the file assigned to the virtio block interface.

### C.2.5 Flash entries

Sysfs Entries:

```
/sys/class/mic/micN/flashversion
/sys/class/mic/micN/flash_update
/sys/class/mic/micN/fail_safe_offset
```

These nodes are all read-only. The `flashversion` sysfs node returns the current version of the flash image installed on the coprocessor by the `micflash` utility. The other two are used by the `micflash` command. Root privileges are required to read `flash_update` and `fail_safe_offset` entries.

### C.2.6 Power management entries

Sysfs Entries:

```
/sys/class/mic/micN/pc3_enabled
/sys/class/mic/micN/pc6_enabled
```

The `pc3_enabled` node reports the current setting of the `pc3` power management setting. If `pc3` power management is causing errors, writing a "0" to this setting will disable `pc3` power management.

The `pc6_enabled` node reports the current setting of the `pc6` power management setting. If `pc6` power management is causing errors, writing a "0" to this setting will disable `pc6` power management.

### C.2.7 Other entries

Sysfs Entries:

```
/sys/class/mic/micN/extended_family
/sys/class/mic/micN/extended_model
/sys/class/mic/micN/fuse_config_rev
/sys/class/mic/micN/meminfo
```
/sys/class/mic/micN/memoryfrequency
/sys/class/mic/micN/memoryvoltage
/sys/class/mic/micN/model
/sys/class/mic/micN/stepping
/sys/class/mic/micN/stepping_data

These sysfs nodes are all read-only and return the contents of a particular hardware register. They are used by the `micinfo` command.
**D  Micrasd**

*micrasd* is a Linux* host side daemon that monitors for and logs Intel® Xeon Phi™ coprocessor hardware errors (MCEs). Normally, *micrasd* is run as a service:

```
[host]# service micras start
[host]# service micras stop
```

To start *micrasd* with secure communications to Reliability Monitor, use:

For RHEL* 6.x and SLES* 11 SP4:

```
[host]# service micras start-with-security
```

For RHEL* 7.x and SLES* 12.x you have to set

```
START_WITH_SECURITY=true on /etc/mpss/micrasrelmond.conf
```

and then start *micras* normally.

The *micras* service has a dependency on the *mpss* service. The *micras* service must be started after the *mpss* service, and stopped prior to stopping the *mpss* service. To automatically start the *micras* service in boot time, use the command:

```
[host]# chkconfig micras on
```

To disable automatically starting the *micras* service, use the command:

```
[host]# chkconfig micras off
```

Coprocessor hardware errors are logged in to Linux* syslog under */var/log/messages* with the *micras* tag.

*micrasd* log messages are logged in to */var/log/micras.log*. These messages can be useful in tracing *micrasd* functional flow for diagnostic purposes.

If *micrasd* is executed with no arguments, it runs at the console prompt, connects to devices, and waits for errors. For more information about *micrasd* refer to:

```
[host]# micrasd -help
```

§
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. If they are found in the default search path (set using the `SINK_LD_LIBRARY_PATH` environment variable), the libraries are copied to the coprocessor prior to execution. This simplifies running coprocessor native applications.

In addition, the utility can also redirect output from an 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.

**Note:** If the application has any library dependencies, then the `SINK_LD_LIBRARY_PATH` environment variable must be set to include those directories. This environment variable works just like `LD_LIBRARY_PATH` for standard Linux* applications. To help determine the required libraries, execute `micnativeloadex` with the `-l` command line option:

```
[host]$ micnativeloadex -l Appname
```

This will display the list of dependencies and which ones have been found. Any dependencies not found will likely need to be included in the `SINK_LD_LIBRARY_PATH`.

Refer to `micnativeloadex` help for more information:

```
[host]$ micnativeloadex -help
```

The `SINK_LD_LIBRARY_PATH` must include the directory path for `libcoi_host.so` library

For example:

```
[host]$ export LD_LIBRARY_PATH=/usr/lib64:$LD_LIBRARY_PATH
```

**Note:** When linking in libraries installed in `/lib64`, do not add `/lib64` to the `LD_LIBRARY_PATH` environment variable. This path is already implicit in the dynamic linker/loader’s search path, and modifying the path variable will result in breaking the order in which library paths are searched for offload compilation.
F Optional Intel® MPSS components

This section provides detailed instructions on installing several optional Intel® MPSS components.

F.1 Intel® MPSS ganglia support

This section describes how to install Ganglia components on the host and the coprocessor for host platforms running RHEL* 6, RHEL* 7 or SLES* 11 SP4. Due to 3rd party incompatibilities Ganglia is not supported on SLES* 12.

F.1.1 Requirements

The following software components must be installed on the host.

1. Red Hat* Enterprise
   - apr
   - apr-devel
   - expat
   - expat-devel
   - gcc-c++
   - libconfuse
   - libconfuse-devel
   - libtool
   - rpm-build
   - rrdtool
   - rrdtool-devel

2. SUSE* Linux* Enterprise Server (SLES*)
   - gcc-c++
   - libapr1
   - libapr1-devel
   - libconfuse0
Optional Intel® MPSS components

- libconfuse-devel
- libexpat0
- libexpat-devel
- libtool
- rpmbuild
- rrdtool
- rrdtool-devel

### F.1.2 Installing ganglia on the host

**Note:** Only GANGLIA 3.1.7 is currently supported.

**Note:** For additional information on the installation of GANGLIA, consult the documentation at [http://ganglia.sourceforge.net](http://ganglia.sourceforge.net)

**Note:** The default path for the GANGLIA web page is `/usr/share/ganglia`. If the ganglia-web RPM was installed, the files `conf.php`, `get_context.php` and `host_view.php` will be overwritten.

#### Steps:

1. Create working directories. For example:
   
   ```
   [host]# mkdir -p /var/lib/ganglia/rrds
   [host]# mkdir -p /var/www/html
   ```

2. Download GANGLIA 3.1.7 from [http://ganglia.info/?p=269](http://ganglia.info/?p=269).

3. Untar GANGLIA 3.1.7 package and access the untar folder:
   
   ```
   [host]$ tar xf ganglia-3.1.7.tar.gz
   [host]$ cd ganglia-3.1.7
   ```

4. Execute the configure tool:
   
   ```
   [host]$ ./configure --with-gmetad \
   --with-libpcre=no --sysconfdir=/etc/ganglia
   ```

5. Build GANGLIA content and install binaries:
   
   ```
   [host]$ make
   [host]$ make install
   ```

6. Generate default configuration for gmond:
   
   ```
   [host]$ gmond --default_config > /etc/ganglia/gmond.conf
   ```
7. Edit (as root) the host's `/etc/ganglia/gmond.conf` and confirm that a `udp_recv_channel` is defined and that it assigns a port value. For example:

```plaintext
udp_recv_channel {
    /*other parameters */
    port = <port>
    /*other parameters */
}
```

If a `udp_recv_channel` is not defined, or if the port is not assigned, then define it. The standard ganglia port is 8649:

```plaintext
udp_recv_channel {
    /*other parameters */
    port = 8649
    /*other parameters */
}
```

8. Edit (as root) the host's `/etc/ganglia/gmetad.conf` to configure the cluster name in the "data_source" line. For example:

```plaintext
data_source "mic_cluster" localhost
```

9. Change the owner of the RRDS folder:

   ```plaintext
   [host]# chown -R nobody /var/lib/ganglia/rrds
   ```

10. Copy GANGLIA web content to local web path.:

    ```plaintext
    [host]# cp -r web <web_path>/ganglia
    ```

11. Start the `gmond` and `gmetad` daemons:

    ```plaintext
    [host]# gmond
    [host]# gmetad
    ```

12. Install web front end for Intel® MPSS GANGLIA.

    - Red Hat* Enterprise Linux*
      ```plaintext
      [host]# yum install $MPSS38/ganglia/mpss-ganglia*.rpm
      ```
    - SUSE* Linux* Enterprise Server
      ```plaintext
      [host]# zypper install $MPSS38/ganglia/mpss-ganglia*.rpm
      ```

13. Copy the web content under `/usr/share/mpss/ganglia` to the GANGLIA web path:

    ```plaintext
    [host]# cp -r /usr/share/mpss/ganglia/* <web_path>/ganglia/
    ```
F.1.3 Installing intel® MPSS ganglia RPMs on the coprocessor

The following RPMs must be installed: ganglia-3.1.7-r0.k1om.rpm, libapr-1-0-1.4.6-r0.k1om.rpm, libconfuse0-2.7-r1.k1om.rpm, and mpss-ganglia-mpss-r0.k1om.rpm

You can use any of the methods described earlier in this section to install Intel® MPSS Ganglia RPMs into the coprocessor file system. In the example below we will use micctrl to add an Overlay RPM parameter for each RPM:

```
[host]# micctrl --overlay=rpm \
--source=$MPSS38_K1OM/libconfuse0-2.7-r1.k1om.rpm --state=on
[host]# micctrl --overlay=rpm \
--source=$MPSS38_K1OM/libapr-1-0-1.4.6-r0.k1om.rpm --state=on
[host]# micctrl --overlay=rpm \
--source=$MPSS38_K1OM/ganglia-3.1.7-r0.k1om.rpm --state=on
[host]# micctrl --overlay=rpm \
--source=$MPSS38_K1OM/mpss-ganglia-mpss-r0.k1om.rpm --state=on
```

Restart the mpss service:

```
[host]# systemctl service mpss restart
```

F.1.4 Starting Intel® MPSS with ganglia support

1. Configure the /etc/ganglia/gmond.conf files on both the host and the coprocessors as needed.

   **Note:** The collection of several CPU metrics is disabled by default in the coprocessor’s /etc/ganglia/gmond.conf. Enabling their collection will cause a performance penalty. To enable these metrics, search for the comment:

   `/*CPU metrics are disabled by default, uncommenting this block will have a performance penalty*/`

   and uncomment the following collection groups.

2. The coprocessor specific GANGLIA stack is started by executing:

   `[host]# ssh mic0 gmond`

F.1.5 Stopping Intel® MPSS with ganglia support

Stop the gmond for all installed coprocessors in the system, for instance:

```
[host]# ssh mic0 killall gmond
[host]# ssh mic1 killall gmond
```

F.2 Performance workloads (micperf)

The coprocessor Performance Workloads component of Intel® MPSS (micperf) can be used to evaluate the performance of coprocessor based installation. The micperf component incorporates a variety of benchmarks into a simple user experience with a
single interface for execution and a unified means of data inspection. The user interface to micperf consists of five executables: one for execution of benchmarks (micprun), and four that interpret the output of the first. These executables are documented with standard UNIX* style command line interfaces. The results can be displayed as professional quality plots, human readable text or comma separated value output that can be easily imported into a variety of other applications. Results of different runs can be easily combined and compared. Documentation is installed at /usr/share/doc/micperf-3.8.

The remainder of this section describes micperf installation.

F.2.1 Installation requirements

1. Intel® Composer XE Requirements:

   There are two options to installing the Intel® Composer XE requirements. The first option is to install the full Intel® Composer XE package and source the compilervars.sh or compilervars.csh script at run time.

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


   This package has an install.sh script for installation. After installation, there are compilervars.sh and compilervars.csh scripts which serve a similar purpose to those scripts in the full Intel® Composer XE distribution and must be sourced at run time.

   Besides the shared object libraries, the MKL Linpack benchmark is also a requirement. This is also freely distributed on the web at:


   This download is a tarball that can be unpacked anywhere, but the environment variable MKLROOT must point to the top level directory of the untarred package. For instance, if the user extracted the tarball into their home directory they should set MKLROOT as follows (in Bash or Bourne shell):

   [host]$ export MKLROOT=<home_directory_path>/linpack_<version_num>

   If MKLROOT is set in the user's shell environment at run time, then micprun will be able to locate the linpack binaries. The version of linpack linked above may be newer than 11.1.2, and MKLROOT variable should reflect this.

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 it only requires MATPLOTLIB for this use case. MATPLOTLIB
should be installed in order to create plots. Download it from: matplotlib.sourceforge.net

F.2.2 Distributed files

This package is distributed as two RPM files:

\texttt{MPSS38/\textasciitilde perf/micperf-3.*.rpm}
\texttt{MPSS38/\textasciitilde perf/micperf-data-3.*.rpm}

The first of these packages contains everything except the reference performance measurements, which are distributed in the second package.

F.2.3 RPM installation

To install the RPM files, \texttt{cd} to \texttt{MPSS38/\textasciitilde perf}, then:

- Red Hat* Enterprise Linux*
  
  \texttt{[host]\# yum install *.rpm}

- SUSE* Linux* Enterprise Server
  
  \texttt{[host]\# zypper install *.rpm}

This installs files to the following directories:

- Source code: \texttt{/usr/src/micperf}
- Documentation and licenses: \texttt{/usr/share/doc/micperf-3.8}
- Benchmark binaries: \texttt{/usr/libexec/micperf}
- Reference data: \texttt{/usr/share/micperf/micp}
- Links to executables: \texttt{/usr/bin}

F.2.4 Python* installation

Once the RPM packages have been installed, an additional step must be executed to access the \texttt{micp} Python package: either install it to your global Python site packages, or set up your environment to use the \texttt{micp} package from the installed location.

To install the package into the Python site packages:

\texttt{[host]\$ cd /usr/src/micperf/micp}
\texttt{[host]\# python setup.py install}

This method provides access to the \texttt{micp} package and executable scripts to all non-root users who use the same Python version as the root user (\texttt{sudoer}). If Python is in the default location and uses a standard configuration, \texttt{setup.py} installs the \texttt{micp} package to the directories:

\texttt{/usr/bin}
Optional Intel® MPSS components

/usr/lib/pythonPYVERSION/site-packages/micp

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

/usr/src/micperf/micp-<version>/build

None of the products of running setup.py discussed above will be removed by uninstalling the micperf RPMs. The installation with setup.py uses Python's distutils module, and this module does not support uninstall. If installing on a Linux* system where Python is configured in a standard way, it should be possible to uninstall with the following commands:

```
[host]# sitepackages=`sudo python -c "from distutils.sysconfig import get_python_lib; print(get_python_lib())"`
[host]# rm -rf /usr/src/micperf/micp/build
/usr/bin/miccsv \\
/usr/bin/micpinfo \\
/usr/bin/micpplot \\
/usr/bin/micpprint \\
/usr/bin/micprun \\
${sitepackages}/micp \\
${sitepackages}/micp-[version number]*
```

F.2.5 Alternative to Python* installation

Another way to access the micp package after installing the RPMs is to alter the shell run time environment of a user. To set up your Bash or Bourne shell environment:

```
[host]# export PYTHONPATH=/usr/src/micperf/micp:${PYTHONPATH)
```

To set up your csh run time environment:

```
[host]# setenv PYTHONPATH /usr/src/micperf/micp:${PYTHONPATH)
```

F.3 Intel® MPSS reliability monitor support

The Intel® MPSS Reliability Monitor is designed to monitor the overall health of compute nodes in a cluster. It typically runs on a cluster’s head, or management, node. The Reliability Monitor works closely with the RAS agent running on each compute node. Uncorrectable errors or crash symptoms are reported to the Reliability Monitor.

F.3.1 Requirements

Intel® MPSS and the micrasd daemon must be installed on each ode to be monitored. micrasd is installed as part of normal software stack installation. Refer to Section 3.4.

F.3.2 Install Intel® MPSS with reliability monitor support

Only install Reliability Monitor on the head node, or management node.

The default path for the Reliability Monitor node configuration file is /etc/mpss.
Steps:

Install Intel® MPSS Reliability Monitor:

- Red Hat* Enterprise Linux*
  
  [host]$ cd $MPSS38/relmon
  [host]$ yum install mpss-sysmgmt-relmon-3.*.rpm

- SUSE* Linux* Enterprise Server
  
  [host]$ cd $MPSS38/relmon
  [host]$ zypper install mpss-sysmgmt-relmon-3.*.rpm

F.3.3 Starting Intel® MPSS with reliability monitor support

1) Make sure that mpss service and micras service are up and running on each compute node. If mpss service and micras service are not running, use:

   [micN]$ # service mpss start
   [micN]$ # service micras start

2) On head node, start Reliability Monitor service by using:

   [host]$ # service relmon start

F.3.4 Stopping Intel® MPSS with reliability monitor support

On head node, stop Reliability Monitor service by using:

   [host]$ # service relmon stop

F.3.5 Reliability monitor configuration file and log

The node configuration file mic_node.cfg for Reliability Monitor is located at /etc/mpss. The file is in comma-separated values (CSV) format so it is supported by almost all spreadsheets and database management systems.

Errors will be logged into Linux* syslog /var/log/messages. You can check the error log using:

   [host]$ # cat /var/log/messages | grep relmon
Reliability Monitor is installed in /usr/bin. After relmon service is running, you can issue commands to monitor node status and error information by using:

```
[host]$ relmond --cmd shownode
[host]$ relmond --cmd showerr
```

For more information about Reliability Monitor, refer to:

```
[host]$ relmond --help
```

§
G Rebuilding Intel® MPSS components

This appendix describes the steps to rebuild selected Intel® MPSS GPL libraries and components. Rebuilding the host and OFED drivers was covered in Sections 3.4.3 and 3.4.5 respectively.

For any of these components, perform the following steps:

1. Install Intel® MPSS (see Section 3.4).
2. Download and untar the mpss-src-3.8.tar file:
   a. Go to the Intel® Developer Zone website (Intel® DZ):
      Download the mpss-src-3.8.tar file from the “SOURCE” link associated with your Intel® MPSS release.
   b. Extract the source archive:
      [host]$ tar xvf mpss-src-3.8.tar
      By definition, source RPMs are extracted to the $MPSS38_SRC directory.

G.1 Recompiling the Intel® MPSS ganglia modules

Support enabled for RHEL* 6 and 7, and SLES* 11 SP4. Ganglia is not supported in SLES* 12.

1. Obtain and install Ganglia prerequisites (see Appendix F.1.1).
2. Obtain and install ganglia-devel-3.1.7 and apr-devel-1.3.9-3 on the host.
   [host]$ cd $MPSS38_SRC
   [host]$ tar xvf mpss-ganglia-mpss.tar.bz2
   [host]$ cd mpss-ganglia-mpss
4. Define the environment variable CROSS_COMPILE.
   [host]$ export CROSS_COMPILE=/opt/mpss/3.8/sysroots/ \ x86_64-mpssdk-linux/usr/bin/kлом-mpss-linux/kлом-mpss-linux
   For RHEL* 7.1 an additional variable is required:
   [host]$ export C_INCLUDE_PATH=/usr/include/apr-1
5. Regenerate the GANGLIA modules.

\[ \text{[host]} \# \text{make} \]

### G.2 Recompiling the Intel® MPSS MIC management modules

1. Install Intel® MPSS™.

2. Ensure that the following packages are installed.
   - `mpss-modules-headers-3.8`
   - `glibc2.12.2pkg-libmicmgmt0-3.8`
   - `libscif0-3.8`
   - `libscif-dev-3.8`
   - `glibc2.12pkg-libmicmgmt-dev-3.8`
   - `asciidoc`

3. Download and untar the `mpss-src-3.8.tar` to `$MPSS38_SRC`

\[ \text{[host]} \# \text{tar xvf mpss-src-3.8.tar} \]


\[ \text{[host]} \# \text{cd $MPSS38_SRC} \]
\[ \text{[host]} \# \text{tar xvf mpss-micmgmt-3.8.tar.bz2} \]
\[ \text{[host]} \# \text{tar xvf mpss-metadata-3.8.tar.bz2} \]

5. Regenerate the Intel® MPSS MIC management modules.

\[ \text{[host]} \# \text{cd $MPSS38_SRC/mpss-micmgmt-3.8} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.mk miclib/} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.c miclib/} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.mk apps/mpssinfo} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.c apps/mpssinfo} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.mk apps/mpssflash} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.c apps/mpssflash} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.mk apps/micsmc} \]
\[ \text{[host]} \# \text{cp ../mpss-metadata-3.8.mpss-metadata.c apps/micsmc} \]

6. Set the DESTDIR environment variable to the desired `make install` target path, for example `/usr/local`.

\[ \text{[host]} \# \text{export DESTDIR=/usr/local} \]
7. Build the `micmgmt` modules:

    [host]# make lib
    [host]# make install_lib
    [host]# make
    [host]# make install

    A build directory will be created at \$DESTDIR, and everything will be installed there.

G.3 How to extract and use the COI open source distribution

COI source is delivered in the file `mpss-coi-3.8.tar.bz2`. In the tar file, the files are packaged with paths relative to the original source directory structure.

G.3.1 Building COI libraries and binaries

1. Ensure that the `asciidoc` utility is installed.

2. Extract `mpss-metadata.tar.bz2` and `mpss-coi-3.8.tar.bz2`:

    [host]$ cd $MPSS38_SRC
    [host]$ tar xvf mpss-metadata-3.8.tar.bz2
    [host]$ tar xvf mpss-coi-3.8.tar.bz2
    [host]$ cd mpss-coi-3.8

3. Rebuild COI, either the debug or release version as needed:

    [host]$ make [debug|release] -I ../mpss-metadata-3.8/

G.3.2 Installing the host library

To install the host-side COI library, first make sure that the Intel® MPSS driver is running, then do the following:

    [host]$ cp build/host-linux-[debug|release]/libcoi_host.so \ 
          /usr/lib64/\ 
    [host]$ cd /usr/lib64/\ 
    [host]$ ln -s libcoi_host.so libcoi_host.so.0

G.3.3 Installing the coprocessor-side binaries and libraries

To install the COI library, first kill the `coi_daemon` so that the new one can be installed:

    [host]$ ssh micN
    [micN]$ killall -9 coi_daemon
    [micN]$ exit

Install the new components and start the COI daemon:

    [host]$ cd $MPSS38_SRC/mpss-coi-3.8
    [host]$ scp build/device-linux-[debug|release]/coi_daemon \ 
          micN:/usr/local/coi-3.8/\ 
    [host]$ ssh micN
    [micN]$ . /usr/local/coi-3.8/coi_daemon
Once installed, and now that the new `coi_daemon` is running, the new COI binaries and libraries will be in use in the current running driver.

Building the COI stack also builds the COI tools. If you wish to install the newly built tools `coitrace` and `micnativeloadex`, do the following:

```
[host]# cp build/host-linux-[debug|release]/coitrace /usr/bin/
[host]# cp build/host-linux-[debug|release]/libcoitracelib.so /usr/lib64/
[host]# cp build/host-linux-[debug|release]/micnativeloadex /usr/bin/
[host]# cd /usr/lib64/
[host]# ln -s libcoitracelib.so libcoitracelib.so.0
```

**G.3.4 Building and running the COI tutorial**

COI tutorials are built using gcc compiler. They are validated on RHEL* 6.3 using gcc 4.4.6. To build and run the COI tutorials, follow the instructions below:

1. Ensure all coprocessors are booted to the **online** state:
   ```
   [host]$ micctrl -s
   mic0: online (mode: linux image: /usr/share/mpss/boot/bzImage-knightscorner)
   ```

   **Note:** To run this command as user in SLES* it is necessary to add the directory `/usr/sbin/micctrl` to user’s `PATH` variable.

2. Extract `mpss-coi-3.8.tar.bz2`:
   ```
   [host]$ cd $MPSS38_SRC
   [host]$ tar xvf mpss-coi-3.8.tar.bz2
   ```

3. Build a COI tutorial, `<coi_tutorial>`:
   ```
   [host]$ cd mpss-coi-3.8/src/tutorial/<coi_tutorial>
   [host]$ make
   ```

4. Execute the debug or release version of the tutorial
   ```
   [host]$ cd [debug|release]
   [host]$ ./<coi_tutorial>_source_host
G.4 How to extract and use the MYO open source distribution

MYO source is delivered in the file `mpss-my0-3.8.tar.bz2`. In the tar file, the files are a tree relative to the `mpss-my0-3.8` directory.

Extract the MYO archive to the desired directory with the following steps.

```
[host]$ cd $MPSS38_SRC
[host]$ tar -xf mpss-my0-3.8.tar.bz2
[host]$ cd mpss-my0-3.8
```

The `mpss-my0-3.8/src/README` text file explains the purpose, content, and use of the MYO Open Source Distribution. It includes information about compiler selection, building and installing the MYO libraries, MYO system requirements, and the MYO tutorials.

MYO tutorials are built using gcc compiler. They are validated on RHEL* 6.3 using gcc 4.4.6.
This section briefly describes how services are started on supported Linux* host Operating Systems and the Intel® Xeon Phi™ coprocessor. This is intended for customers who are adding custom services that may interact with the services supplied with Intel® MPSS. In all cases the described priority only applies to initial boot and run level changes. The priority or dependencies are not checked when services are manually started and stopped.

H.1 Service startup by priorities (RHEL* 6.x)

Red Hat* traditionally uses this method of startup and shutdown. A line is added to the top of the service script that defines the run levels and priority of when a service starts at boot time.

Here is an example snippet from the top of a service file:

```
#!/bin/bash
# chkconfig:  23 45 10 90
# ...
```

This tells the startup daemon in Linux* to shut down the service early (priority 10) and start the service late (priority 90) when entering Linux* run levels 2, 3, 4, and 5. If a service A depends on another service B the shutdown and startup priorities should reflect the relative prioritie sooner and later respectively:

```
#!/bin/bash
# Service B
# chkconfig:     2345 10 90

and:

#!/bin/bash
# Service A
# chkconfig:     2345 9 91
```

The priority increases in time for both shut down and startup of a service. Now service A will start after service B and service B will shut down after service A.

When you have multiple dependencies make sure the new service’s shutdown time is the minimum of the dependencies minus 1 and the start priority is max of dependencies + 1.

There is a tool for managing services runlevels and priorities called `chkconfig`. For more details please see:

In addition to the chkconfig comment line from the Red Hat* distribution priority method, SUSE* Linux* Enterprise Server 11 adds a new concept to the startup order, dependencies. The chkconfig method is present for backward compatibility.

Here is a snippet we can refer to:

```bash
#!/bin/bash
# chkconfig: 35 75 54
# Description: Novell Identity Manager User Application
### BEGIN INIT INFO
# Provides: userapp
# Required-Start: $ndsd $network $time
# Required-Stop:
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Short-Description: Novell IDM UserApp
# Description: Novell Identity Manager User Application
### END INIT INFO
```

Some short definitions:

**Provides**  - The name used to identify this service in the init daemon

**Required-Start**  - Space delimited Provides names of services to start before this service

**Required-Stop**  - Space delimited Provides names of services to stop before this service

**Default-Start**  - Space delimited list of run levels to start when transitioning run levels

**Default-Stop**  - Space delimited list of run levels to stop when transitioning run levels

**Short-Description**  - Short display name of service

**Description**  - Full display name of service

To make sure the service start order is correct, pick the list of service dependencies and list them on the Required-Start line. Make sure to fill in the start and stop run levels as appropriate. Optionally list the services to stop after the service represented by this script.

**Note:** All names used for service reference must be the Provides name and not the file name of the script.

For more details on this method see:
H.3 Service start priority on the coprocessor

The coprocessor’s init daemon using the SUSE® Linux® Enterprise Server 11 dependency system is described in the previous section.

§
I Troubleshooting and debugging

This appendix is a collection of tips and techniques that can be helpful in troubleshooting an Intel® Xeon Phi™ coprocessor installation and/or coprocessor debugging execution.

I.1 Log files

The coprocessor supports BusyBox* implementations of dmesg and syslogd.

I.1.1 Dmesg output

Viewing dmesg output can sometimes help in troubleshooting when a coprocessor fails to boot.

First, verify if debugfs is mounted:

```
[host]$ mount | grep debugfs
none on /sys/kernel/debug type debugfs (rw)
```

Mount debugfs, if not already mounted:

```
[host]$ mount -t debugfs none /sys/kernel/debug
```

Coprocessor dmesg output can be viewed during coprocessor boot (or later) at /sys/kernel/debug/mic_debug/micN/log_buf. For example:

```
[host]$ cat /sys/kernel/debug/mic_debug/micN/log_buf
```

I.1.2 Syslog output

By default, each coprocessor’s syslog messages are logged to the coprocessor’s /var/log/messages file. The log target and other logging details, such as the log (severity) level, can be changed using the micctrl --syslog command:

```
micctrl --syslog=(buffer|file|remote) \
    [--host=<targethost[:port]>] [--logfile=<location>] \
    [--loglevel=<loglevel>] [mic card list]
```

Of particular note is that syslog messages can be forwarded to the host or another node (when the coprocessor is bridged to the external network). For this purpose, the targethost syslog or rsyslog daemon must be configured for UDP reception on the specified port. On RHEL* 6, uncomment the following line in /etc/rsyslog.conf:

```
# Provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514
```

For example:

```
# Provides UDP syslog reception
$ModLoad imudp
```

$UDPServerRun 514

The syslog or rsyslog daemon then typically must be restarted in order to pick up this new configuration. On RHEL* 6, the rsyslog daemon is restarted as follows:

[host]# /etc/rc.d/init.d/rsyslog restart

If a host firewall is enabled, it may need to be configured to allow forwarding of syslog messages to the specified host. By default, the syslog or rsyslog daemon listens on UDP port 514. Consult your firewall documentation for configuration help.

Now use the *micctrl --syslog=remote* command to, for example:

[host]# micctrl --syslog=remote

In this case the *<targethost[:port]>* defaults to *host:514*. See *Appendix B.4.7.5* for more details on the *micctrl --syslog* command.

If not using *micctrl* (configuring manually), edit the */etc/syslog-startup.conf* file in the default ramfs image. Consult BusyBox* documentation on the parameters in this configuration file.

## I.2 Coprocessor post codes

Like any other Intel® IA-32, Intel® 64 or IA-64 platform, the coprocessor produces POST codes at power on and boot to identify the stage that the card is at during the boot process. These POST codes can be viewed using the Linux* command "dmesg" after a system power on. The POST codes can also be viewed during a boot cycle of the coprocessor by "tailing" */var/log/messages*:

[host]# tail -f /var/log/messages | grep "Post Code"

The current POST code of a coprocessor can be obtained from its sysfs node:

[host]$ cat /sys/class/mic/micN/post_code

The POST codes are defined as follow:

"01" LIIDT
"02" SBOX initialization
"03" Set GDDR top
"04" Begin memory test
"05" Program E820 table
"06" Initialize DBOX
"09" Enable caching
"0b" Pass initialization parameters to APs
"0c" Cache C code
"0d" Program MP Table
"0E" Copy AP boot code to GDDR
"0F" Wake up APs
"10" Wait for APs to boot
"11" Signal host to download coprocessor OS
"12" Wait for coprocessor OS download - this is also known as the "ready" state. The coprocessor will be in this state after powering on, running "micctrl -r" or "service mpss stop". It means that the coprocessor is ready to receive the coprocessor OS either by a "service mpss start", "service mpss restart" or "micctrl -b" depending on how the coprocessor got into this state. It is not an error condition for the coprocessor to be in this state. See the sections above to learn how to start Intel® MPSS when the coprocessor is showing POST code 12
"13" Signal received from host to boot coprocessor OS
"15" Report platform information
"17" Page table setup
"30" Begin memory training
"31" Begin GDDR training to query memory modules
"32" Find GDDR training parameters in flash
"33" Begin GDDR MMIO training
"34" Begin GDDR RCOMP training
"35" Begin GDDR DCC disable training
"36" Begin GDDR HCK training
"37" Begin GDDR ucode training
"38" Begin GDDR vendor specific training
"39" Begin GDDR address training
"3A" Begin GDDR memory module identification
"3b" Begin GDDR WCK training
"3C" Begin GDDR read training with CDR enabled
"3D" Begin GDDR read training with CDR disabled
"3E" Begin GDDR write training
"3F"  Finalize GDDR training
"40"  Begin coprocessor OS authentication
"50"-"5F"  coprocessor OS loading and setup
"6P"  int 13 General Protection
"75"  int 10 Invalid TSS
"87"  int 16 x87 FPU Floating Point Error
"AC"  int 17 Alignment Check
"bP"  int 3 Breakpoint
"br"  int 5 BOUND Range Exceeded
"CC"  int 18 Machine Check
"co"  int 9 coprocessor Segment Overrun
"db"  int 1 Debug
"dE"  int 0 Divide Error
"dF"  int 8 Double Fault
"EE"  Memory test failed
"F0"  GDDR parameters not found in flash
"F1"  GBOX PLL lock failure
"F2"  GDDR failed memory training
"F3"  GDDR memory module query failed
"F4"  Memory preservation failure
"F5"  int 12 Stack Fault
"FF"  Bootstrap finished execution
"FP"  int 19 SIMD Floating Point
"Ld"  Locking down hardware access
"nA"  coprocessor OS image failed authentication
"nd"  int 7 Device Not Available
"no"  int 2 Non-maskable Interrupt
"nP"  int 11 Segment Not Present
"oF"  int 4 Overflow
"PF"  int 14 Page fault
"r5"  int 15 reserved
"ud"  int 6 Invalid opcode

I.3  Installing Intel® MPSS debug information

The $MPSS38/dbg directory contains debug packages for several software stack components. The packages can be installed by following the instructions below.

- RHEL*:

  [host]# cd $MPSS38/dbg
  [host]# yum install *.rpm

- SLES*:

  [host]# cd $MPSS38/dbg
  [host]# zypper install *.rpm

The /usr/lib64/.debug, /usr/bin/.debug and /usr/sbin/.debug directories contain the debug symbols.

I.4  Kernel crash dump support

1. The host driver configuration option to enable/disable coprocessor kernel crash dumps is located in /etc/modprobe.d/mic.conf.

   # crash_dump enables Coprocessor OS Kernel Crash Dump Captures
   # 1 to enable or 0 to disable
   : options mic reg_cache=1 huge_page=1 watchdog=1
   watchdog_auto_reboot=1 crash_dump=1 p2p=1 p2p_proxy=1 ulimit=0

   Crash dump support is enabled by default. Edit the options line to disable support.

2. The mpssd daemon configuration options to tune crash dump storage location and storage limit (gigabytes) are typically in the /etc/mpss/default.conf Intel® MPSS configuration file.

   # Storage location and size for MIC kernel crash dumps
   CrashDump /var/crash/mic/ 16

   Edit the CrashDump parameter to change the crash dump storage location and limit.

3. If a coprocessor OS crash occurs, a gzipped kernel crash dump core file will be available at the storage location configured in step 2.
4. Install the crash utility on the host to analyze the crash dump (RHEL* example shown):

   [host]# yum install crash

5. An example showing how a crash dump can be analyzed is shown below:

   [host]$ cd /var/crash/mic/mic0/
   [host]$ gunzip vmcore-xxxx.gz
   [host]$ cp /opt/mpss/3.8/sysroots/klo-mpss-linux/boot/vmlinux-2.6.38.8+mpss3.8
   [host]$ /opt/mpss/3.8/sysroots/klo-mpss-linux/boot/x86_64-klo-linux-elfedit --output-mach x86_64 vmlinux-2.6.38.8+mpss3.8
   [host]$ crash vmlinux-2.6.38.8+mpss3.8 vmcore-2012-9-24-15:50:29

   Useful commands include foreach, bt, ps, log, etc.

   Refer to http://people.redhat.com/anderson/crash_whitepaper/#HELP

6. If a custom user space utility other than the mpssd daemon is being used, then a crash dump can be obtained as follows:

   a) Poll the sysfs entry /sys/devices/virtual/mic/ctrl/subsystem/mic0/state for coprocessor state changes.

   b) Upon detection of the "lost" state, read from /proc/mic_vmcore/ and write the contents to a crash dump file.

   c) Gzip the content of the file.

   d) Now reset the card and reboot it if required.

I.5 Gnu Debugger (gdb) for the coprocessor

GDB can be used to debug applications on a coprocessor. GDB supports both native execution on a coprocessor as well as remote execution from a host processor. The Debugging with GDB manual is installed as the file $MPSS38/docs/GDB.pdf; it provides detailed instructions on the use of GDB. This section presents some additional information on using GDB on the coprocessors.

I.5.1 Running gdb natively on the coprocessor

To execute GDB natively, the RPM file $MPSS38_K1OM/gdb-7.4+mpss3.8-1.k1om.rpm must be installed into the coprocessor file system. Refer to Section 7 for help on installing RPMs into the coprocessor file system.

I.5.2 Running remote gdb on the coprocessor

The remote coprocessor enabled GDB client is located on the host at:
The GDB Server is pre-installed in the coprocessor file system by default at:

```
/usr/bin/gdbserver
```

For complete GDB remote debugging instructions, refer to the section “Debugging Remote Programs” in the GDB manual.

### I.5.3 Gdb remote support for data race detection

GDB supports data race detection based on Intel® PDBX data race detector for Intel® Many Integrated Core (MIC) architecture. See the "Debugging data races" section in the GDB manual.

Ensure that the environment is set up correctly and that GDB finds the correct version of the Intel® compiler’s run-time support libraries. See the PROBLEMS-INTEL file in the GDB source package for additional help on troubleshooting.

### I.5.4 Debugging heterogeneous/offload applications

Heterogeneous application debugging is supported in Eclipse*. This requires the installation of an Eclipse* plugin. Install `mpss-eclipse-cdt-mpm-*x86_64.rpm`.

Installation steps for the Eclipse* plugin:

1. From the Eclipse* menu use "Help" -> "Install new Software".
2. Click on "Add".
3. Click on "Local".
4. Use the `/usr/share/eclipse/mic_plugin` path and click "OK".
5. Click "OK" again in the popup window.
6. Unselect the following two checkboxes: "Group items by category" and "Contact all update sites during install".
7. Select the plugin using the corresponding checkbox, then click "Next".
8. Click "Next".
9. Accept the license agreement and click "Finish".
10. In the “Security Warning” popup, click “OK”.
11. Restart the Eclipse* IDE.
I.5.5 Enabling mic gdb debugging for offload processes

An environment variable must be set in order to allow the debugger to enable module name mapping with the generated files needed to attach to the coprocessor side offload processes. To do this, execute the following step:

```
[host]$ export AMPLXE_COI_DEBUG_SUPPORT=TRUE
```

§