Intel® Optane™ Persistent Memory

Start Up Guide

October 2020

Revision 2.0

Contents

1 Getting Started

Intel® Optane™ persistent memory represents a new class of memory and storage technology architected specifically for data center usage. Platforms based on the 2nd Gen and 3rd Gen Intel® Xeon® Scalable processors may be populated with a combination of DRAM and Intel® Optane™ persistent memory.

This document will use "PMem" in reference to the Intel® Optane™ persistent memory module.

1.1 Basics

Intel® Optane™ Persistent Memory modules support two modes: Memory Mode, which is volatile, and App Direct mode, which is byte addressable persistent memory. The modes determine which capabilities of the Intel persistent memory module are active and available to software. With the Intel® Optane™ PMem 100 series the two operating modes may be configured to run concurrently.

1.1.1 Operating Modes

In **Memory Mode**, the DRAM acts as a cache for the most frequently accessed data, while the Intel® Optane™ persistent memory provides large memory capacity. Cache management operations are handled by the Intel® Xeon® Scalable processor's integrated memory controller.

When data is requested from memory, the memory controller first checks the DRAM cache, and if the data is present, the response latency is identical to DRAM. If the data is not in the DRAM cache, it is read from the Intel® Optane™ persistent memory with slightly longer latency.

The applications with consistent data retrieval patterns that the memory controller can predict will have a higher cache hit-rate, and should see its performance close to all-DRAM configurations, while workloads with highly-random data access over a wide address range may see some performance difference versus DRAM alone.

Data is volatile in Memory Mode; it will not be saved in the event of power loss. Persistence is enabled in the second mode, called App Direct.

In **App Direct Mode**, applications and the Operating System are explicitly aware there are two types of direct load/store memory in the platform and can direct which type of data read or write is suitable for DRAM or Intel® Optane™ persistent memory.

Operations that require the lowest latency and don't need permanent data storage can be executed on DRAM, such as database "scratch pads". Data that needs to be made persistent or structures that are very large can be routed to the Intel® Optane™ persistent memory. To make data persistent in memory, you must use App Direct Mode.

App Direct mode requires an operating system or virtualization environment enabled with a persistent memory-aware file system. Contact your OS distributor for support details.

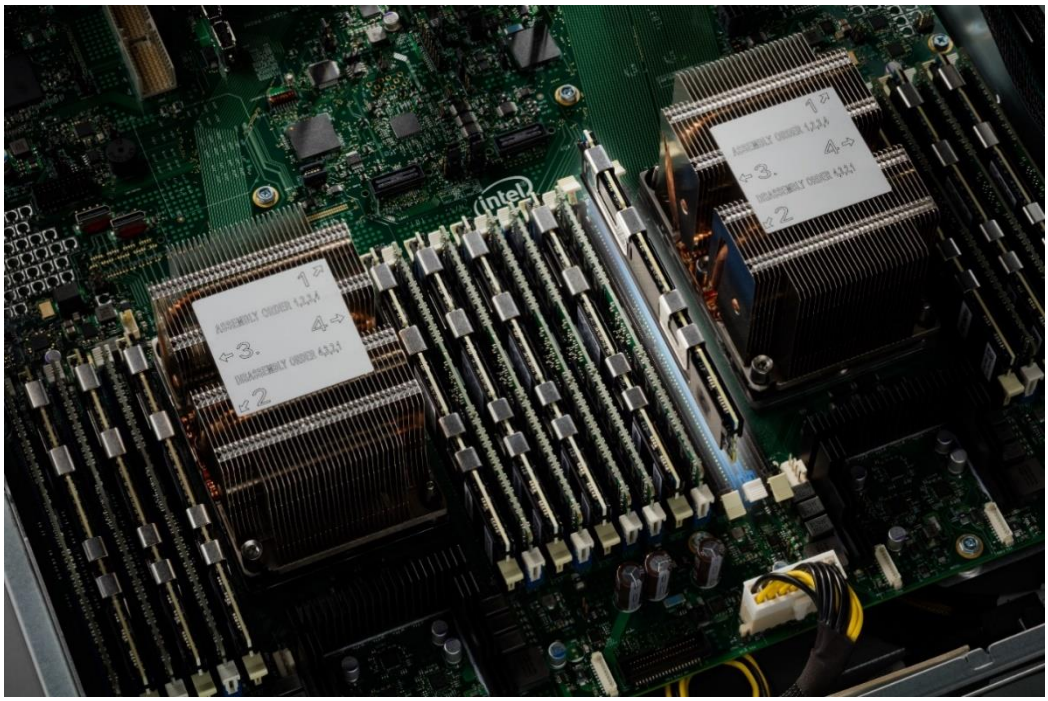
1.1.2 Using Persistent Memory

App Direct Mode may be used as Storage over App Direct, in which case, the driver surfaces a traditional block storage interface transparent to applications, so they do not need to be modified. Storage over App Direct is implemented with copy on write optimization via a block translation table to provide power-fail write atomicity.

Otherwise, applications can be modified to access App Direct capacity with direct load/store mechanisms using a persistent memory aware file system. This completely bypasses the kernel and provides the shortest code path to the persistent memory. To learn more about using and programming for persistent memory, refer to http://pmem.io/.

Warning: A backup of the persistent memory data sets to secondary storage is needed so that the backup may be used to restore the original data sets after a data loss event.

1.2 Installation



1.2.1 System Requirements

Intel® Optane™ persistent memory is designed for use with the 2nd and 3rd Gen Intel® Xeon® Scalable Processors family and is enabled by a sub-set of processors, primarily enabled by the Gold and Platinum tier models.

The Intel® Optane™ PMem 200 series is enabled with 3rd Gen Intel® Xeon® Scalable Processors. The Intel® Optane™ PMem 100 series is enabled with the 2nd Gen Intel® Xeon® Scalable Processors. PMem modules are not enabled in any other hardware combinations. Mixing PMem generations or product capacity SKUs within a system is not supported. The DDR4 and provisioned PMem module's memory configuration shall be uniformly configured and provisioned across all sockets.

Table 1 PMem and Processor Compatibility

Intel® Optane™ Persistent Memory	**Intel Xeon Scalable Processors**
Intel® Optane™ PMem 100 series 	2nd Gen Intel® Xeon® Scalable Processors
Intel® Optane™ PMem 200 series 	3rd Gen Intel® Xeon® Scalable Processors

1.2.1.1 Intel Optane PMem 200 Series

The Intel PMem 200 series is enabled on four socket (4S) platforms with the 3rd Gen Intel® Xeon® Scalable Processors.

Workloads are optimized for the App Direct persistent memory mode. Memory mode cannot be enabled on this platform and processor. The memory frequency speed with installed PMem population is 2666 MT/s. See also Section Recommended Topologies.

1.2.2 Software and Firmware Requirements

Note: Ensure to update to the latest SW/FW published by the OEM the platform of choice.

Platform software and Intel® Optane™ persistent memory firmware alignment is strongly recommended. The PMem active (operating) firmware version for all installed modules must be the same prior to normal/runtime operations.

The benefits of updating the software and firmware is to keep the platform software current and compatible with other system modules. Updates may include changes such as fixed issues, improved stability and security updates.

1.2.3 Handling, Installation and Removal

Industry standard DIMM practices and procedures must be followed while handling, installing or removing PMem. While taking proper ESD (Electrostatic Discharge) measures care must be taken not to stack modules together and to handle only by the module edges, never with enough force to flex or bend the module.

Modules removed and populated in another supported system must be re-provisioned to reconfigure the PMem module. See Section 2.3 Maintenance.

1.2.4 DIMM Populations

1.2.4.1 Recommended Topologies

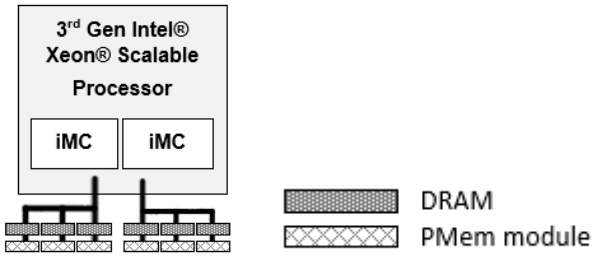
The following topologies are recommended per CPU socket. For multiple socket systems, each socket should be populated identically. The part number must be the same for all installed PMem. Populating both slots on a single channel with PMem is not supported.

Note: DDR channel and DIMM slot nomenclatures may vary depending on platform implementation. Guidance for PMem installation in first position (slot 0) followed by the DIMM (slot 1). When one DIMM is used, it must be populated furthest away from the CPU (slot 0) of the channel.

1.2.4.2 Intel Optane PMem 200 Series Topology

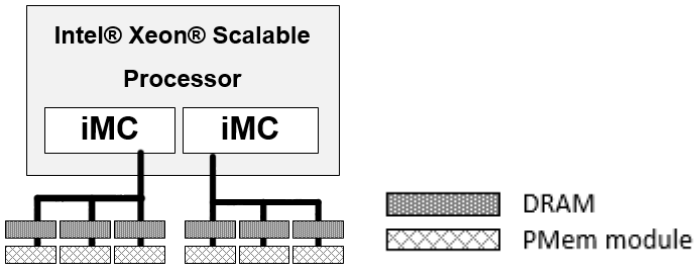
Note: Memory configuration enforcement and/or BIOS option settings may have been adopted and implemented in the OEM vendor UEFI FW (BIOS) that may issue warning or may even prevent boot. Contact the vendor for memory population guidance and BIOS behavior and/or option settings support information.

6+6 (Six DDR4 DIMMs, Six PMem)
Supported Mode: App Direct Only
Memory frequency: 2666 MT/s

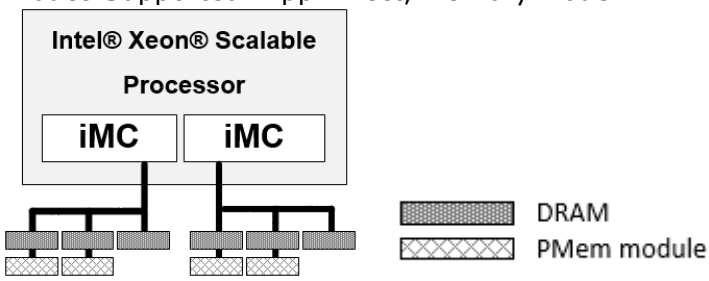


1.2.4.3 Intel Optane PMem 100 Series Topologies

6+6 (Six DDR4 DIMMs, Six PMem)
Modes Supported: App Direct, Memory Mode

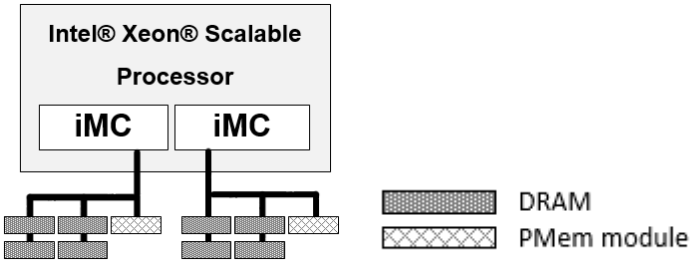


6+4 (Six DDR4 DIMMs, Four PMem)
Modes Supported: App Direct, Memory Mode



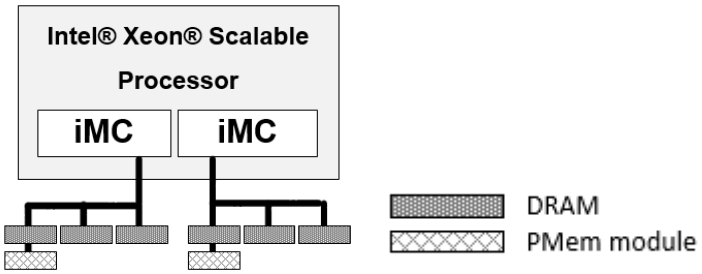
8+4 (Eight DDR4 DIMMs, Two PMem)

Modes Supported: App Direct, Memory Mode



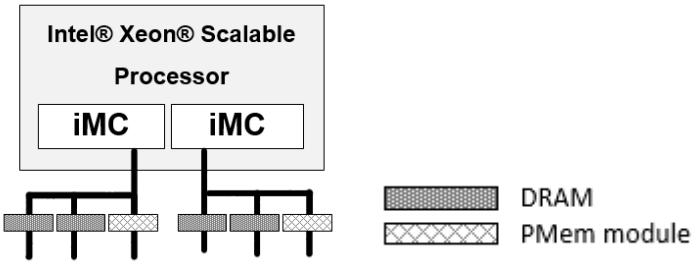
6+2 (Six DDR4 DIMMs, Two PMem)

Modes Supported: App Direct, Memory Mode



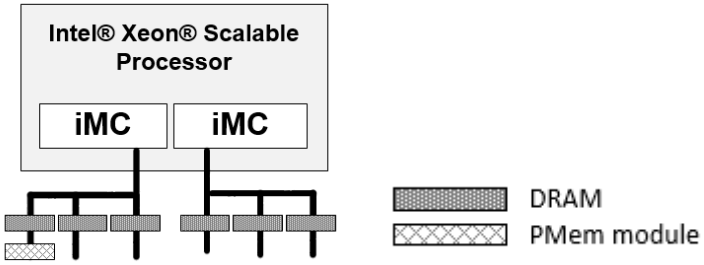
4+2 (Four DDR4 DIMMs, Two PMem)

Modes Supported: App Direct, Memory Mode



6+1 (Six DDR4 DIMMs, One PMem)

Modes Supported: App Direct Only



1.2.5 Documentation and Software Resources

Intel article (Linux* and Windows* Provisioning Examples): https://software.intel.com/content/www/us/en/develop/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux.html

Non-Volatile Device Control (ndctl) is recommended for creating namespaces for the Linux operating system. It is available for download from GitHub* at https://github.com/pmem/ndctl.

To learn more about persistent memory programming and the Persistent Memory Development Kit (PMDK), visit: http://pmem.io/.

Ipmctl is an open source utility for configuring and managing PMem. This utility was created and is maintained by Intel. It is available for download from GitHub*. at https://github.com/intel/ipmctl.

2 Managing PMem

Intel Ipmctl is a platform utility for configuration and management capabilities including discovery, provisioning, maintenance, and monitoring of the PMem modules.

It supports the following functionality.

- Discovery
- Configuration
- Firmware management
- Security functionality management
- Health monitoring
- Performance tracking
- Debug and troubleshooting

For ipmctl details on release branches and its target PMem module refer to the ipmctl GitHub* project.

Note: This document provides examples of essential PMem management functionality from the perspective of the OS ipmctl command line utility.

```
ipmctl help
```

Example commands are formatted where [brackets] indicate optional items and (parenthesis) indicate that a value is required. The binary executable name "ipmctl" is omitted for simplicity.

2.1 Discovery

The *Show Topology* command displays both the PMem and DDR4 DRAM DIMMs.

```
ipmctl show -topology
```

The *Show Device* command displays the PMem discovered in the system.

```
ipmctl show -dimm
```

To view detailed information about the PMem, add the all option.

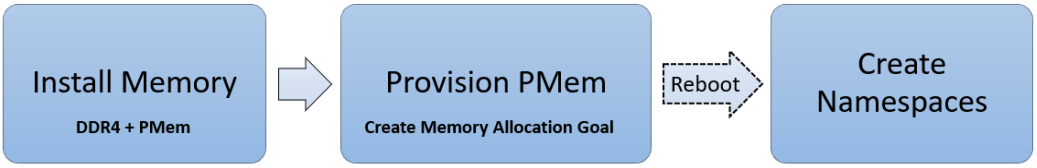
```
ipmctl show -a -dimm
```

The *Show Memory Resources* command displays how the DDR/PMem capacity is allocated at the system level.

```
ipmctl show -memoryresources
```

2.2 Provisioning

Provisioning Intel® Optane™ PMem is a two-step process. During this process, a goal is specified and stored on the persistent memory modules for the BIOS to read on the next reboot. A goal configures the PMem in Memory Mode, App Direct mode. App Direct persistent memory regions may then be provisioned with namespaces which the driver surfaces as a logical device.



The Intel Optane PMem 100 series may be configured to operate both modes concurrently. Platforms with Intel Optane PMem 200 series cannot be configured for concurrent modes. Memory Mode cannot be configured with the 3rd Gen Intel® Xeon® Scalable processors.

The installed DIMM population topology also impacts which modes are supported; refer to the recommended topology figures previously shown.

Warning: Changing the memory configuration is a destructive operation which may result in the loss of data stored in the persistent memory region of the PMem. Therefore, existing data should be backed up to other storage prior to provisioning if it needs to be preserved.

Persistent data is not explicitly cleared during provisioning. Metadata such as file system and SW RAID should be deleted prior to provisioning. The PMem module may be erased by destroying the cryptographic persistent memory key and/or overwriting. Sanitize methods may be implemented in UEFI environment by the OEM vendor. The Linux OS runtime environment provides the ndctl-sanitize-command https://pmem.io/ndctl/ndctl-sanitize-dimm.html. This command is rejected (unable) to sanitize the PMem module if Memory Mode is configured.

2.2.1 Create Memory Allocation Goal

Use the *Create Memory Allocation Goal* command to create a new configuration request.

Memory Mode 100%

Note: Memory Mode cannot be configured for the Intel Optane PMem 200 series for platforms with the 3rd Gen Intel® Xeon® Scalable processors.

Any percentage of the PMem capacity across sockets can be provisioned in Memory Mode. In this example, **100** represents the percentage of capacity to be provisioned in Memory Mode.

```
ipmctl create -goal MemoryMode=100
```

App Direct Mode

PMem can be provisioned in App Direct mode with the interleaving enabled or disabled for the persistent memory.

App Direct (Interleaved)

Interleaving increases the throughput of reads and writes to persistent memory. The default *Create Goal* command creates an interleaved region configured for App Direct mode. The following two commands are equivalent:

```
ipmctl create -goal

ipmctl create -goal PersistentMemoryType=AppDirect
```

App Direct (Not Interleaved)

To disable interleave, keep the App Direct capacity contained to each PMem, specify PersistentMemoryType=AppDirectNotInterleaved.

```
ipmctl create -goal PersistentMemoryType=AppDirectNotInterleaved
```

Concurrent Operating Modes (Mixed Mode)

Note: Mixed Mode cannot be configured for the Intel Optane PMem 200 series with the 3rd Gen Intel® Xeon® Scalable processors.

The following command assigns 60 percent of the available persistent memory capacity to Memory Mode. The remainder is configured as an interleaved set for App Direct mode.

```
ipmctl create -goal MemoryMode=60
```

The *Show Memory Allocation Goal* command displays a pending goal request prior to reboot. In the case that a goal is not applied correctly, this command returns the configuration request and status. "No result" is returned if the goal was processed successfully during reboot.

```
ipmctl show -goal
```

The *Delete Memory Allocation Goal* command enables the removal of a pending goal.

```
ipmctl delete -goal
```

2.2.1.1 Advanced Preferences

The *APPDIRECT_GRANULARITY* preference allows for adjusting the default settings when creating a memory allocation goal. The default granularity for Intel Optane PMem 200 series 1GiB. Therefore, the preference setting has been removed in the ipmctl 2.00.00.xxxx release.

To view the current preferences, use the *Show Preferences* command.

```
ipmctl show -preferences
```

To change preferences, use the *Change Preferences* command.

```
ipmctl set -preferences (Name=Value)
```

Preference Name	Description	Options
APPDIRECT_GRANULARITY	The minimum App Direct granularity per PMem.	RECOMMENDED: Use the default recommended App Direct granularity of 32 GiB for Intel Optane PMem 100 series. 1: Allow 1 GiB App Direct granularity.

2.2.2 Persistent Memory and Namespace Management

Each OS vendor (OSV) provides native tools for persistent memory and namespace management. Contact the OSV/ISV for more information.

Intel article (Linux* and Windows* examples):
https://software.intel.com/content/www/us/en/develop/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux.html

The following sections provide Linux examples and references for detailed explanation.

Linux References:

The official project documentation for ndctl.
http://pmem.io/ndctl/

The official project location for ndctl.
https://github.com/pmem/ndctl

Basic guide to getting started with persistent memory in Linux.
https://nvdimm.wiki.kernel.org/

2.2.2.1 Linux Command Line (ndctl) Overview

The ndctl is a utility library for managing the libnvdimm (non-volatile memory device) sub-system in the Linux kernel. This library does not provide functionality for memory allocation provisioning, the ipmctl create -goal equivalent functionality.

The ndctl command line provides functions used for persistent memory and namespace management, device list, update firmware and more.

2.2.2.2 Listing Regions

Man page: https://pmem.io/ndctl/ndctl-list.html

After the memory provisioning (goal processing) reboot, the newly created DIMM-interleave-sets are represented as persistent memory "regions" of App Direct capacity. Regions can be listed with the *List Regions* command.

```
# ndctl list --regions --human
```

Sample output:


```
[
  {
    "dev":"region1",
    "size":"756.00 GiB (811.75 GB)",
    "available_size":"756.00 GiB (811.75 GB)",
    "max_available_extent":"756.00 GiB (811.75 GB)",
    "type":"pmem",
    "iset_id":"0x6f3e7f4888992ccc",
    "persistence_domain":"memory_controller"
  },
  {
    "dev":"region0",
    "size":"756.00 GiB (811.75 GB)",
    "available_size":"756.00 GiB (811.75 GB)",
    "max_available_extent":"756.00 GiB (811.75 GB)",
    "type":"pmem",
    "iset_id":"0x1c9e7f487b952ccc",
    "persistence_domain":"memory_controller"
  }
]
```


To determine which PMem are part of a particular region, use the *list* command with the *regions* and *dimms* modifiers. Each PMem will be listed within the *mappings* group.

```
# ndctl list --regions --dimms
```

Sample output:


```
{
  "dimms":[
    {
      "dev":"nmem1",
      "id":"8680-a2-1730-000006a5",
      "handle":257,
      "phys_id":45
    },
    {
      "dev":"nmem3",
      "id":"8680-a2-1730-00000632",
      "handle":4353,
      "phys_id":69
    },
    {
      "dev":"nmem0",
      "id":"8680-a2-1730-0000059f",
      "handle":1,
      "phys_id":33
    },
    {
      "dev":"nmem2",
      "id":"8680-a2-1730-000004f3",
      "handle":4097,
      "phys_id":57
```

```
    }
  ],
  "regions":[
    {
      "dev":"region5",
      "size":268435456000,
      "available_size":268435456000,
      "max_available_extent":268435456000,

      "type":"pmem",
      "iset_id":8743142107817513552,
      "mappings":[
        {
          "dimm":"nmem3",
          "offset":268435456,
          "length":134217728000,
          "position":1
        },
        {
          "dimm":"nmem2",
          "offset":268435456,
          "length":134217728000,
          "position":0
        }
      ]
    },
    {
      "dev":"region4",
      "size":268435456000,
      "available_size":0,
      "max_available_extent":0,
      "type":"pmem",
      "iset_id":8015984336272174284,
      "mappings":[
        {
          "dimm":"nmem1",
          "offset":268435456,
          "length":134217728000,
          "position":1
        },
        {
          "dimm":"nmem0",
          "offset":268435456,
          "length":134217728000,
          "position":0
        }
      ]
      “persistence_domain”:”memory_controller”
    }
  ]
}
```

2.2.2.3 Creating Namespaces

Man page: https://pmem.io/ndctl/ndctl-create-namespace.html

Regions can then be divided into one or more namespaces in order for the capacity to be surfaced to the Linux operating system and used by applications. Just as an SSD can be carved into namespaces, persistent memory namespaces represent the unit of storage that appears as a device that can be used for I/O. To create a namespace, use the *Create Namespace* command.

```
# ndctl create-namespace
```

The default syntax will create namespace with DAX support, mode=*fsdax.* A DAX aware filesystem may subsequently be created on this namespace to provide optimal application performance. The following example is equivalent syntax to explicitly set the mode and also provides the optional syntax to specify a target region.

```
# ndctl create-namespace –-mode fsdax [--region (value)]
```

To create a Storage over App Direct namespace for use as a traditional block storage device with power-fail write atomicity, use sector mode.

```
# ndctl create-namespace –-mode sector [--region (value)]
```

Multiple namespaces may be created from the same region. Each namespace is a minimum of 1 GiB (in bytes) and aligns to interleave-width and alignment; for example, '*--size=*' must align to *interleave-width: 6* and *alignment: 2097152*. Sector namespaces may then be further subdivided by a partition table.

To create multiple namespaces of specific capacity within a single region, use the *Create Namespace* command with the *Size* option. The ---continue option will create as many namespaces as possible within the given -bus and -region filter restrictions.

2.2.2.4 Listing Namespaces

Man page: https://pmem.io/ndctl/ndctl-list.html

To display namespaces, use the *List* command. Namespaces are shown by default without additional arguments.

```
# ndctl list --human
```

Sample output:

```
[
  {
    "dev":"namespace1.0",
    "mode":"fsdax",
    "map":"dev",
    "size":"744.19 GiB (799.06 GB)",
    "uuid":"56ca2d82-36fc-4bf6-b434-b5315957f5ca",
    "blockdev":"pmem1"
  },
  {
    "dev":"namespace0.0",
    "mode":"fsdax",
    "map":"dev",
    "size":"744.19 GiB (799.06 GB)",
    "uuid":"5ec652fa-3642-4e93-ad76-90d70872f3a3",
    "blockdev":"pmem0"
  }
]
```

2.2.2.5 Delete Configuration

The current configuration can be deleted by first disabling and destroying namespaces and then disabling the active regions. When a namespace is destroyed, the capacity is returned to the underlying region and may be used in the creation of new namespaces.

2.3 Maintenance

This section describes some common maintenance operations.

2.3.1 Firmware Update

The *Show Device Firmware* command using ipmctl.

```
ipmctl show -firmware
```

This command displays up to two firmware versions per PMem considering that updating the firmware requires a power cycle. The active firmware is the firmware that is currently executing. The "staged" firmware is stored on the PMem and staged for execution after the next power cycle.

If for any reason the firmware update fails to load properly after the power cycle, the firmware will fall back to the previously active version.

Note: Only one firmware is allowed to be staged per power cycle. Therefore, if a version is already staged, power cycle before attempting to stage a new version.

The *Update Firmware* command provides an examine option which validates a firmware image file and may be used prior to loading the image.

```
ipmctl load -source (firmware file) -examine -dimm
```

Load the firmware image on the PMem using the *Update Firmware* command. It is recommended that all PMem in the system have the same firmware version, although updating the firmware on individual PMem is also supported.

```
ipmctl load -source (firmware file) -dimm
```

In case the PMem firmware is busy processing a long-operation command request, such as Address Range Scrub (ARS), it will not be possible to update the firmware and an error will be returned.

Check that the new firmware was staged properly using the *Show Device Firmware* command and then power cycle the system. Before using the PMem, verify that the new firmware is now successfully executing on all PMem using the *Show Device Firmware* command.

Linux Workarounds for ARS Operation

To wait for completion of an active Linux ARS long operation(s) and then proceed with the *Update Firmware* command, use the following command:

```
ndctl wait-scrub; ipmctl load -source (firmware file) -dimm
```

Note: ARS may take hours to days depending on the persistent memory configuration and capacity. In such case, it may be preferable to disable the ARS at boot via the kernel boot command line parameter.

The *disable ARS process* command will require adding a kernel boot parameter, rebooting the OS and then the firmware update. After verifying the firmware updated successfully, ensure to remove the added kernel boot command entry to restore the original (default) ARS state setting and execution of ARS operations.

1. Add *Kernel Boot Command Line Entry*:

   ```
   nfit.no_init_ars
   ```

2. Reboot Linux.
3. Verify that an ARS long operation(s) is not currently active.

   ```
   ipmctl show -d ARSStatus -dimm
   ```

4. Stage the firmware, using the *Update Firmware* command.

   ```
   ipmctl load -source (firmware file) -dimm
   ```

5. Check that the new firmware was staged properly using the *Show Device Firmware* command and then power cycle the system.

```
ipmctl show -dimm -firmware
```

6. Verify a successful firmware update using the *Show Device Firmware* command.
7. Remove the "nfit.no_init_ars" kernel boot command line entry and reboot the OS in order to restore the default Linux background ARS operation setting.

2.3.2 Adding New PMem

Adding new PMem module(s) to a system that is already configured provides for a few choices depending on the current configuration, the platform settings, and the desired usage of the new PMem.

New PMem will be used in Memory Mode if the platform supports it. The newly added PMem can be configured for use in a different mode leaving the existing PMem as they were. Or the new and existing PMem can be configured for use in a different mode together.

Refer to the provisioning section for detailed steps.

2.3.3 Moving PMem

The information that describes how the PMem are provisioned and interleaved together along with the namespace information is physically stored on the PMem. Therefore, it is possible to move PMem from one system to another and retain the configuration and any data stored in the persistent memory.

However, if the PMem are configured with interleaved App Direct Mode capacity, the PMem must be put into the new system so they can be interleaved in the same way as they were in the previous system. Therefore, it is necessary to install them in an identical system, ensure that the same position relative to the CPU, memory controller, and memory channel is maintained. UEFI FW (BIOS) version and settings duplicate that of the originating system platform.

PMem modules that are configured for use in only Memory Mode or App Direct Mode that is not interleaved can be installed in any order in the new system and the configuration and data stored on the PMem will be applied.

2.3.4 Replacing PMem

Any App Direct capacity that is interleaved requires that all of the PMem modules in the interleave set (generally all the PMem installed on a specific CPU) are present and functional in order to access the persistent data. Therefore, if it becomes necessary to replace a PMem, further actions are required to maintain the persistent data before replacing the PMem.

Before removing the PMem, the data must be backed up to other storage and all existing namespaces must be deleted. If the same memory configuration is desired after replacing the PMem, dump the current configuration to a file using the *Dump Memory Allocation Settings* command.

```
ipmctl dump -destination (file) -system -config
```

Then replace the PMem and load the configuration as a new memory allocation goal using the *Load Memory Allocation Settings* command.

```
ipmctl load -source (file) -goal
```

Because this creates a new memory allocation goal, a reboot is necessary to map the capacity into the system physical address space. After the reboot, proceed with creating namespaces as desired and restoring the existing data to the new namespaces.

2.4 Debug and Troubleshooting

This section provides information about some common operations that may be required to debug or troubleshoot an issue with a PMem.

2.4.1 Checking PMem Health

The *Show Device* command return data includes the "HealthState" and HealthStateReason attributes.

```
ipmctl show -a -dimm
```

Display select attributes with a comma separated list.

```
ipmctl show -d HealthState,HealthStateReason -dimm
```

For additional health statistics, use the *Show Sensor* command.

```
ipmctl show -a -sensor
```

Sensors are reported for the following health data.

Sensor Name	Description
Health	The current PMem health as reported in the SMART log
MediaTemperature	The current PMem media temperature in Celsius
ControllerTemperature	The current PMem controller temperature in Celsius
PercentageRemaining	Percentage of life remaining
LatchedDirtyShutdownCount	The number of shutdowns without notification over the lifetime of the PMem
UnlatchedDirtyShutdownCount	The number of shutdowns without notification over the lifetime of the PMem. This counter is the same as LatchedDirtyShutdownCount, except it will always be incremented on a dirty shutdown even if Latch System Shutdown Status was not enabled

Sensor Name	Description
PowerOnTime	The total power-on time over the lifetime of the PMem
UpTime	The total power-on time since the last power cycle of the PMem
PowerCycles	The number of power cycles over the lifetime of the PMem
FwErrorCount	The total number of firmware error log entries

PMem support alarm thresholds for the percentage value of expected life span remaining, the temperature of the Intel® Optane™ Media, and the temperature of the controller. These are customizable using the *Change Sensor Settings* command.

```
ipmctl set -sensor (sensor) NonCriticalThreshold=(value)
EnabledState=(0|1)
```

If enabled, the PMem will send an alarm when the reading exceeds the threshold and the sensor status will change to a *non-critical* state.

2.4.2 Diagnostics

Beyond checking the health information, diagnostics are provided to perform more detailed checks of the PMem using the *Run Diagnostic* command.

```
ipmctl start -diagnostic
```

2.4.3 Recovering from Issues

As PMem modules near the end of their lifespan or encounter issues, recovery actions may be required to bring the module and the overall system back to a functional state. Refer to the following sections to determine how best to handle different types of issues based on the current state.

Table 2 Issue Categories

Issue	How to tell
Temperature	ipmctl show -a -sensor MediaTemperature ipmctl show -a -sensor ControllerTemperature
Noncritical	ipmctl show -dimm: HealthState = Noncritical
Critical	ipmctl show -dimm: HealthState = Critical
Fatal	ipmctl show -dimm: HealthState = Fatal
Non-functional	ipmctl show -dimm: HealthState = Non-functional
Busy	ipmctl show -dimm: ARSStatus = In progress ipmctl show -dimm OverwriteStatus=In progress
Unmanageable	ipmctl show -dimm: HealthState = Unmanageable
Setup issue	ipmctl show -dimm: ConfigurationStatus = Failed
Population issue	ipmctl show -dimm: PopulationViolation = Yes

Issue	How to tell
Missing namespace	ipmctl show -namespace or native OS NVDIMM command/tool

2.4.4 Temperature

Note: Environmental effects such as changes to Media and Controller temperature do not change the firmware SMART Health Status (HS). A temperature alarm trip may exist in any SMART Health Status (HS) or ipmctl HealthState.

When nearing the temperature alarm threshold, the module should be cooled immediately to avoid further remediation such as throttling or shutdown. In the event of temperature exceeding an alarm threshold an alert is triggered (if enabled).

```
ipmctl show -a -sensor MediaTemperature

ipmctl show -a -sensor ControllerTemperature
```

2.4.5 Noncritical

There are two indications that a PMem module may be at risk.

1. The firmware SMART health status changes to *Noncritical.* When this happens, an alert is triggered (if enabled).
2. The ipmctl reported HealthState is *Noncritical*.

When Noncritical state occurs, investigate to identify possible causes.

Review Health State/Status Reason(s)

```
ipmctl show -d HealthStateReason -dimm
```

Table 3 Noncritical HealthStatusReason

Meaning	Maintenance Action
0% < Percentage Remaining ≤ 1%	Remaining lifespan is minimal. Plan maintenance to backup persistent data and replace the impacted module.
Package Sparing has happened	Informational only, no remedial maintenance action is available. This HSR bit will remain set for the remaining life of the module. Continue to monitor the module for secondary reasons associated with the Noncritical health state, including monitoring for percentage remaining.
CAP Self-Test[1] returns a warning and last test was Normal	Plan maintenance to backup persistent data and replace the impacted module.
CAP Self-Test communication failure	Before power cycle, backup persistent data and power cycle to determine if self-test communication is restored.
[1] CAP Self-Test warning means the Power Loss Capacitor has degraded to low levels but is not in a state that will result in data loss.	

2.4.6 Critical

When the firmware SMART health status changes to *Critical*, features or performance are degraded due to a failure and it is at risk of data loss/failure. This is indicated in one of the following ways.

1. The firmware SMART health status changes to critical. When this happens, an alert is triggered (if enabled).
2. The ipmctl reported HealthState is *Critical*.

When *Critical* state occurs, investigate to identify possible causes.

Review Health State/Status Reason(s)

```
ipmctl show -d HealthStateReason -dimm
```

Table 4 Critical HealthStatusReason

Meaning	Maintenance Action
Percentage Remaining 0	Before power cycle, backup persistent data and replace the impacted module.
Die Failure	
AIT DRAM failure occurred	
CAP Self-Test fails[1]	
Performance Degraded	
CAP Self-Test communication failure	*Before* power cycle, backup persistent data and power cycle to determine if self-test communication is restored. If issue persists, replace the impacted module.
[1] CAP Self-Test fails means the Power Loss Capacitor has degraded, this state can result in data loss on the next power cycle.	

2.4.7 Fatal

When a PMem module fails, any persistent data stored on the module is inaccessible and the module must be replaced or removed in order to restore overall system functionality.

When *Fatal* state occurs, communication may still be possible via SMBus to retrieve status reason and/or logs.

1. The firmware SMART health status changes to *Fatal*. When this happens, no alert is triggered because the DDRT link is taken down by firmware.
2. The ipmctl reported HealthState is *Fatal*.

Review Health State Reason(s)

```
ipmctl show -d HealthStateReason -dimm
```

Table 5 Fatal HealthStatusReason

Meaning	Maintenance Action
Critical internal state failure[1]	If possible, backup persistent data and replace the impacted module. Persistent data is typically no longer accessible.
[1] Critical internal state failure is non-recoverable and data loss has occurred or is imminent.	

2.4.8 Non-functional

Persistent data is not accessible in this state.

When the ipmctl HealthState is Non-functional, the PMem module is detected and manageable, though some commands and capabilities may be limited. The PMem module has limited communication or another error preventing complete functionality. Common causes include:

- DDRT memory interface training failure
- Expected region mapping to SPA range unable to be found

Some ipmctl commands may be able to communicate with the module via SMBus.

When Non-functional state occurs, investigate to identify possible causes:

1. A power cycle may recover from select PMem firmware errors.
2. Possible UEFI FW memory margining DDRT training errors may be found by examining the system initialization BIOS serial log.
3. Review the firmware BootStatus:

```
ipmctl show -d BootStatus -dimm <DimmID>
```

   - **Media Not Ready:** firmware did not complete media training
   - **Media Error:** firmware detected an error during media training
   - **Media Disabled:** firmware disabled media due to a critical issue
   - **FW Assert:** firmware reported an assert during initialization

2.4.9 Busy

When a long operation such as an Address Range Scrub (ARS) or security overwrite is running, some Intel® Optane™ PMem functionality may be limited. Once the long operation is complete, the functionality will be restored.

How to tell:

```
ipmctl show -d ARSStatus -dimm

ipmctl show -d OverwriteStatus -dimm
```

2.4.10 Unmanageable

Manageability is the ability for the host software to manage a PMem module. An *Unmanageable* HealthState can occur when the PMem module has an incompatible firmware API version or hardware revision or is unresponsive.

If the Intel® Optane™ PMem firmware is incompatible with the software, refer to the Software Installation section to install a compatible version of the firmware and software.

Manageability is determined by the interface format code, the vendor identifier, device identifier and the firmware API version. Running the quick health diagnostic may help identify the reason for the incompatibility.

If the vendor or device information is incorrect or the firmware API version is unavailable, and the device is a PMem module, it is likely the module is not responding. This could occur at runtime, for example, if the PMem module is shut down due to a thermal issue. Cooling down the PMem module and restarting may resolve the *Unmanageable* state. If the module is not responding due to a hardware failure, the module will need to be replaced or removed in order to restore overall system functionality.

2.4.11 Setup Issue

Provisioning is an interaction between the software, the platform firmware, and metadata stored on the PMem modules during a system reboot. While unlikely, it is possible that an incompatibility exists, or a hardware failure occurs that results in a configuration status issue as reported in the *Show Device* command. Run the platform configuration diagnostic to get more details about the cause of the failure.

If the issue is the result of old metadata stored on the PMem modules, re-provisioning the modules may fix the issue. If it is necessary to recover/override the Platform Configuration Data refer to the ipmctl Delete Platform Configuration Data command.

If the issue is the result of a PMem module failure, refer to the preceding sections to determine the correct actions.

2.4.12 Population Issue

The topology or memory population is expected to align to one of the documented POR memory configurations for DDR4 and Intel® Optane™ PMem modules. Contact the server platform vendor for memory population guidance.

2.4.13 Missing Namespace

Persistent memory namespaces are the unit of storage that appear as a logical device to the OS which can be used for I/O. One or more missing namespaces could be caused by the following.

1. The Linux software is waiting for an address range scrub long operation to complete before surfacing the namespaces to the OS. In this case, wait for the long operation to complete before attempting to use the namespace.
2. One or more PMem modules contributing capacity to the namespace are security locked with a passphrase. The namespaces will not be surfaced to the OS until the underlying modules are unlocked or security is disabled by removing the passphrase.
3. One or more PMem modules contributing capacity to the namespace has encountered an issue and the persistent memory capacity is not available. Refer to the preceding sections to determine the correct action.

2.4.14 Dump Debug Log

Occasionally, PMem encounter an error that requires debug information to be gathered for offline analysis. This can be done using the *Dump Debug Log* command. This debug log is also referred to as the *Firmware Debug Log*.

```
dump -destination (file) -debug -dimm (DimmID)
```

The *Firmware Debug Log* is in binary form and it is not encrypted or signed.

2.4.15 Dump System Support Data

The *Dump Support Data* command provides functionality to create and save platform level support data to a file for off-line analysis by support personnel.

Support data includes system log(s), error log(s), sensor information and diagnostic results.

```
dump -destination (file) -support
```