# ODM and Install Requirements for IHV to Enable New Fibre Channel or iSCSI Device in AIX

Version 2.5
January 18, 2019

## Table of Contents

# 1 Document Control

Version:Version 1.0, August 26, 2005: Initial Revision

Version 1.1, Feb 23, 2009: Minor Updates

Version 1.2, Jan 20, 2011: Minor Updates.

Version 2.0 August 6, 2014: Major updates; remove text copied from AIX publications; add information about MPIO ODM, reformat other information

Version 2.1 October 27, 2015: Updates related to concurrently changeable attributes.

Version 2.2 August 03, 2016: Updates related to Logical Block Provisioning

Version 2.3 June 06, 2017: Add description of LUR_on_error attribute

Version 2.4 June 11, 2018: Updates for iSCSI MPIO, path_isid attribute. Update for max_retries attribute.

Version 2.5 January 18, 2019: Update rw_timeout with statement about minimum value.

# 2 Abstract

This document describes the ODM database entries, message file, and package that vendors are required to provide to enable support for their Fibre Channel or iSCSI storage devices in AIX. It also explains how to create a user installable package. This document is to be used as a guide to help vendors create their ODM files, their message file, and the install package containing them.

# 3 Related Documentation

The AIX publications contain additional information about many of the concepts and commands used in this book. Publications for current versions of AIX can be found online in the IBM Knowledge Center:

Those publications can be searched for more information about any of the AIX commands that this document references, as well as for more general information about the ODM databases that are used by the AIX MPIO software.

# 4 Terminology

The following terminology is used in this document:

- bff or BFF - Backup File Format. A bff (or BFF) image is an installation image file created in a backup file format that contains a package and its associated fileset(s). AIX installp packages are provided as BFF images.
- config methods - AIX configuration methods. Utilities used to configure AIX devices.
- Dynamic Tracking – An AIX facility that allows seamless recovery from changes to the FC SAN that result in changes to the N_Port ID (aka SCSI ID) of remote storage ports.
- LBP – Logical Block Provisioning. Also known as “Thin Provisioning”.
- MPIO – Multi Path I/O. An AIX facility that allows using multiple paths between the operating system and the storage device.
- ODM - Onbect Data Manager. Database containing information describing all devices on an AIX system.
- PCM - Path Control Module
- PdAt - Predefined Attribute. An ODM object class.
- PdAtXtd - Predefined Attribute Extended. An ODM object class.
- PdPathAt -- Predefined Path attribute ODM object class
- PdDv - Predefined Devices . An ODM object class.
- SP - Service Pack. A collection of fixes that are applied to a Technology Level. Often abbreviated SP. For example, AIX 5.3TL6 SP8.
- TL – Technology Level. A collection of fixes applied to a version of AIX. Often abbreviated TL. For example, AIX 6.1 TL9.
- UDID - Unique Device IDentifier
- VRMF - Version Release Modification Fixlevel. The product version number.

# 5 Introduction

In AIX, device discovery and configuration is not initiated in the kernel, but rather, it is driven by user level utilities referred to as config (configuration) methods. The config methods for Fibre Channel or iSCSI devices initiate device discovery by sending inquiries to the device and comparing the results with data that has been previously stored in a database on the system. This database contains information describing all devices for the system, and is referred to as ODM (Object Data Manager).

This document is to be used as a guide to help vendors enable support for a new Fibre Channel or iSCSI device types on AIX by creating ODM files, a message file, and an install package that will be installed by the customer to enable the new device type support. Note that in AIX, FCoE (Fibre Channel over Ethernet) adapter device drivers are implemented with the same interface as the traditional FC (Fibre Channel) adapters. So in this document, any reference to "FC" or "Fibre Channel" actually encompasses both traditional Fibre Channel and Fiber Channel over Ethernet.

This document begins with a discussion of high level requirements. The ODM section follows. It starts with an overview of ODM, a description of ODM commands, and an overview of the ODM requirements to support a new device type. This is followed by individual sections addressing PdDv, PdAt, and PdAtXtd ODM entries, required and optional attributes, and optional features.

After ODM, the message catalog is described, followed by the section covering install and packaging. Useful Commands and Frequently Asked Questions sections follow.

Prior to version 2.0 of this document, only the ODM that was required for defining a device with a single path (i.e. a non-MPIO device) was described. For many years, AIX has supported configuring multiple paths to Fibre Channel devices under a single hdisk instance, but all information about MPIO support was contained in a separate PCM toolkit. Version 2.0 of this document now includes some of the MPIO information, which may allow enabling a device for MPIO. See the MPIO section of this document for additional details

Multiple paths are supported for the iSCSI Software Initiator starting with AIX 7.2 TL3. On earlier levels of AIX, an iSCSI device may still be recognized as an "MPIO" device but the iSCSI protocol driver will only allow a user to configure one path to that device.

Please note a couple of important points from the disclaimer after the title page. This document is provided on an "AS IS" basis, with no guarantee of support and no obligation from IBM. This document must not be redistributed to anyone outside of the organization that IBM provides the document to. IBM does not have a formal support or qualification process for third party vendors who use this document to attach their devices to AIX.

# 6 Requirements

Below is a list of what vendors must provide to add a new device type to an AIX system:

- One or two PdDv entries

- Multiple PdAt entries
- Multiple PdAtXtd entries (required only to support using webSM to manage the devices)
- Message catalog file
- Install package containing the above

Note that some features discussed in this document have been added to AIX in the last few years. To support one of these features, minimum AIX levels are required. When appropriate, this document will identify required levels of AIX for a given feature. Any feature that does not have specific levels of AIX identified is supported on all currently supported levels of AIX.

The remainder of this document addresses each of these requirements in detail.

# 7 Before you Start

Before creating ODM for a device, an assessment should be completed to determine if the storage device is a good fit for support on AIX. The AIX storage stack places some requirements on devices that may not be present in other initiators, and the AIX storage stack implements its own subset of SCSI standard features. The following questions will help you determine if the device will work well with AIX.

1. Does the device require the use of any vendor specific SCSI commands, or does the device require processing of any vendor specific sense data? If either of these are true, then the existing AIX storage stack may not work with the device as it only implements SCSI standard commands and sense data. A device that uses proprietary commands or sense data may require a dedicated PCM.

2. Does the device provide support for the NACA bit set to 1 in the SCSI CDB, and provide support for the ACA task attribute? This support is indicated by the "NormACA" bit in the standard inquiry data being set to 1. AIX will queue multiple commands only to devices that support NACA set to 1. If the device does not support NACA set to 1, it will still function with AIX, but will be limited to a queue depth of 1 (i.e. 1 command outstanding at a time). This may result in poor performance.

3. Does the device use Asynchronous Logical Unit Access (ALUA) as indicated by the TPGS bits being non-zero in the standard inquiry data? AIX will recognize ALUA devices and use the optimized paths when possible. However, AIX does not implement the "Set Target Port Group" command (i.e. explicit ALUA). Also, AIX may pause I/O as long as any port group is in "Transitioning" state, so if the device indicates transitioning state, it should be only for brief periods of time.

4. Does the device implement the SCSI WRITE AND VERIFY(10) and WRITE AND VERIFY(16) commands? These commands are listed as "Optional" commands in the SCSI Block Commands Standard. However, the AIX Logical Volume Manager (LVM) requires WRITE AND VERIFY. So any attempt to create a volume group will fail if the device does not support these commands.

When you are ready to create your install package, you will need to make sure you have the bos.adt.insttools package installed on your machine. To determine if you already have the package, run the following command:

```
lslpp -l |bos.adt.insttools
```

If you are missing the package, you need to install it before proceeding. The bos.adt.insttools package has a prerequisite of perl.rte 5.6. So to install bos.adt.insttools, you will need access to the AIX install media that has the bos.adt.insttools and perl.rte.5.6 or higher packages

Once it is installed, you should find the mkinstallp utility on your system under /usr/sbin.

## 7.1 Gathering Information

When you are ready to create your package, the process will go quicker if you have already decided on the following information. Detailed descriptions are provided in the Install subsection.

- copyright information
- package name
- product version number (VRMF)

## 7.2 MPIO Support

The AIX operating system configures disk devices in ODM using the prefix "hdisk" followed by an integer number to differentiate the different disks. Each LUN on the storage device may have one or more hdisks associated with it in AIX, as follows:

- For a "non-MPIO" configuration, AIX creates one hdisk to represent each path to the given LUN. This configuration is useful for some third party multi-path solutions that are available on AIX. It may also be used in environments where redundancy is not important and there is only one physical path between AIX and the storage device.

- For "MPIO" configuration, AIX creates only one hdisk to represent the LUN on the storage device, with separate path entries to represent the physical paths to the device. An MPIO configuration requires a "Path Control Module" (PCM) to provide AIX with algorithms for selecting paths and processing error conditions.

AIX also includes a "default PCM", which may work with devices that do not require any proprietary handling of error sense codes or other error conditions. This default PCM is designed to work with active/active devices (where all ports on the storage device are treated equally) or with devices that support Asynchronous Logical Unit Access (ALUA). This document includes a description of the ODM required to configure an MPIO device using the default PCM, which provides a minimal effort method of supporting multiple paths.

Note: IBM makes no guarantees that the AIX default PCM will work with any particular third party device other than those explicitly supported by AIX, nor does the default PCM support all parts of any version of the SCSI standards. It is the storage vendor's responsibility to verify if their device and the AIX default PCM function well together.

## 7.3 Thin Provisioning Support

Starting version 7.2 TL1 onwards, AIX shall support any storage that provides thin-provisioning (a.k.a Logical Block Provisioning) capabilities if **all** of the following conditions are met:

1. Support SCSI Inquiry 0xB0 (Block Limits VPD Page) and provide LBP related information such as “Optimal Unmap Granularity”, “Unmap Granularity Alignment”, “Maximum Write Same Length”

2. Support SCSI Inquiry 0xB2 (Logical Block Provisioning VPD Page) and include fields such as “LBPWS”, “LBPRZ”, “Provisioning Type”.

3. Support SCSI command read_capacity(16) and provide LBP related information such as “LBPME” and “LBPRZ”.

4. SCSI Inquiry page 0x0 should include 0xB0 and 0xB2 as supported inquiry pages.

5. Support SCSI write_same(16) command with unmap option.

6. Define a new PdAt ODM Attribute named “lbp_enabled = 'true'” for the respective storage.

Note that AIX will issue a request to unmap disk blocks (via write_same(16)) only for those disks that have the “Provisioning Type” field in Inquiry 0xB2 set to “010b”.

# 8 ODM

This section starts with an overview of ODM, a description of ODM commands, and an overview of the ODM requirements to support a new device type. This is followed by individual sections addressing the ODM classes PdDv, PdAt, and PdAtXtd. Under each PdDv, PdAt, and PdAtXtd section, detailed syntax and field descriptions, a list of required and optional attributes, and feature specific attributes are given where applicable. Descriptions of how to create your vendor pecific stanza files are also included. This section ends with a description of how to test your ODM *.odmadd file.

## 8.1 ODM Overview

ODM (Object Data Manager) is the database stored on an AIX system that describes all supported devices, as well as other system information. It is used by the config methods to discover and configure devices. In addition, it provides persistent device naming across reboots. If new storage is attached to the system, ODM prevents device name slippage, even if the new devices are discovered first.

ODM is composed of object classes and objects. Information is stored and maintained as objects with associated characteristics. Throughout this document ODM objects are referred to as ODM entries.

You add entries into the ODM database by defining the entries in an ASCII file in stanza format. The file's name should have .odmadd as the suffix. The entries are then added to the ODM database using the odmadd command, which takes the ASCII file as input.

The names of the ODM classes that are related to device discovery and naming all start with either "Pd" or "Cu". "Pd" indicates a "predefined" class. Predefined classes contain information about potential devices that the system may recognize and configure. "Cu" stands for "Customized", and contains information about the devices that have actually been discovered by the running AIX instance. So, for example, the "Pd" databases may tell AIX how to recognize storage devices from storage vendors A, B, and C. The AIX configuration methods use those entries to identify devices that are discovered to be physically attached to the AIX instance, and then the configuration methods create "Cu" entries to represent the discovered devices. So if only storage vendor B's device is attached to this instance and there are 10 LUNs defined on that device, then after configuration, the "Cu" databases will contain information about 10 disk devices. Generally, the "Pd" entries are static and the "Cu" entries are created by the AIX configuration methods.

The relevant ODM databases are identified and briefly described below.

PdDv - Predefined Devices

> The Predefined Devices (PdDv) object class contains entries for all device types that the AIX instance could recognize.

PdAt - Predefined Attribute

> The Predefined Attribute (PdAt) object class contains an entry for each existing attribute for each device represented in the Predefined Devices (PdDv) object class

PdAtXtd - Predefined Attribute Extended

> The Predefined Attribute Extended (PdAtXtd) object class is used to supplement existing device's attributes represented in the Predefined Attribute (PdAt) object class with information that can be used by Device Management User Interface.

CuDv - Customized Devices

> The Customized Devices (CuDv) object class contains entries for all device instances defined in the system.

CuAt - Customized Attribute

> The Customized Attribute (CuAt) object class contains customized device-specific attribute information. There will be a CuAt entry for any attribute that does not have the default value for any device.

## 8.2 ODM Commands

Multiple ODM commands that allow the database to be queried and manipulated are provided on an AIX host. They allow the administrator to create, retrieve, modify, and delete ODM database entries. A description of these commands can be found in the man pages or in the AIX publications.

The ODM commands are:

| | |
|---|---|
| odmadd | Adds objects to an object class. The odmadd command takes an ASCII stanza file as input and populates object classes with objects found in the stanza file. |
| odmget | Retrieves objects from object classes and puts the object information into odmadd command format. |
| odmdelete | Removes objects from an object class. |
| odmchange | Changes specific objects in a specified object class. |
| odmcreate | Creates empty object classes. The odmcreate command takes an ASCII file describing object classes as input and produces C language .h and .c files to be used by the application accessing objects in those object classes. |
| odmdrop | Removes an entire object class. |
| odmshow | Displays the description of an object class. The odmshow command takes an object class name as input and puts the object class information into odmcreate command format. |

These commands may be used during testing and development of the ODM files. But once the ODM files are packaged into an AIX install image, the appropriate ODM commands are executed as part of the installation process.

Warning: Incorrect use of the ODM commands may result in an unusable system. For example, accidentally deleting key entries from PdDv may prevent AIX from recognizing and configuring the boot device. It is highly recommended that any experimentation with the ODM commands be done only on a test system for which you have a recent backup or appropriate AIX installation media to re-install the system if the ODM becomes unusable.

## 8.3 ODM Requirements for New Device Types

To ensure AIX is able to take advantage of a device's unique attributes, the vendor must provide ODM entries that describe their specific devices. Three types of entries are required to be added to the Predefined database when adding a new device type. They are:

- One or two PdDv entries
- Multiple PdAt entries
- Mutliple PdAtXtd entries (required only for using webSM to manage devices).

To discover and configure LUNs, ODM compares returned inquiry data to PdDv entries. If a vendor specific PdDv entry has not been provided, a device will still be configured, but it will be assigned a generic disk type, which AIX references as type "Other" when displaying disk information with the lsdev command.

To ensure AIX takes advantage of a device's unique attributes, vendors need to provide a PdDv entry that matches their device and multiple PdAt and PdAtXtd entries describing their device's attributes.

## 8.4 PdDv (Predefined Devices)

The Predefined Devices (PdDv) object class contains entries for all device types that the system is able to recognize.

A PdDv object is required to add a new device type to AIX.  Each device type, as determined by class-subclass-type information, is represented by an object in the PdDv object class. These objects contain basic information about the devices, such as device method names and instructions for accessing information contained in other object classes. This entry is used to identify the device's class, subclass, and device type. AIX uses a hierarchical system to represent devices.  A device is defined by

- Functional classes
- Functional subclasses
- Device types

For more information on device classification, see the "Device Classes" section in the AIX publications.

You build a Predefined Device object by defining the objects in an ASCII file in stanza format. The PdDv entry is then added to the ODM database using the odmadd command, which takes the ASCII file as input.

The format of a PdDv entry is shown below, using the "Other FC SCSI Disk Drive" PdDv as an example. This is followed by a brief description of each field.

This section ends with a description of how to create your PdDv entry.

### 8.4.1 PdDv Format and Example

An example PdDv entry is shown below, followed by a description of the PdDv fields.  This example is the PdDv entry for the default "Other FC SCSI Disk Drive" device. Not all fields are not required in any given PdDv entry in the ASCII stanza file; any field omitted from the ASCII file will take on a default value (blank for string fields and 0 for numeric fields).

Not all fields in the PdDv database are relevant for storage devices.  Unused or NULL string fields are set to "" (two double-quotation marks with no separating space) and unused integer fields are set to 0 (zero).

```
PdDv:
  type = osdisk
  class = disk
  subclass = fcp
  prefix = hdisk
  devid = ""
  base = 1
  has_vpd = 1
  detectable = 1
  chgstatus = 0
  bus_ext = 0
  led = 0x626
  setno = 2
  msgno = 70
  fru = 1
  catalog = scdisk.cat
  DvDr = scsidisk
  Define = /usr/lib/methods/define
  Configure = /usr/lib/methods/cfgscsidisk
  Change = /usr/lib/methods/chgdisk
  Unconfigure = /usr/lib/methods/ucfgdevice
  Undefine = /usr/lib/methods/undefine
  Start = ""
  Stop = ""
  uniquetype = disk/fcp/osdisk
```

A brief description of the PdDv fields is provided below. A full description of the PdDv database and the fields it contains may be found in the AIX publications.

| | |
|---|---|
| type | Device type Specifies the product name or model number. For example, osdisk above. You would put your vendor disk type here. |
| class | Specifies the functional class name. This should be "disk" for disk devices. |
| subclass | Identifies the device subclass associated with the device type. For disk devices, the subclass is the protocol used to attach to the device, either "fcp" or "iscsi" or "sas". |
| prefix | Specifies the Assigned Prefix in the Customized database, which is used to derive the device instance name and /dev name. For disks, this is usually "hdisk". |
| devid | Device ID. Not used for disks. |
| base | Flag indicating a base device. A base device is any device that forms part of a minimal base system. |
| has_vpd | Specifies whether device instances belonging to the device type contain extractable vital product data (VPD). |
| detectable | Specifies whether the device instance is detectable or nondetectable. |
| chgstatus | Indicates the initial value of the Change Status flag used in the Customized Devices (CuDv) object class. |
| bus_ext | Indicates that the device is a bus extender. |
| inventory_only | Distinguishes devices that are represented solely for their replacement algorithm from those that actually manage the system. |
| fru | Identifies the type of field replaceable unit (FRU) for the device. |

| | |
|---|---|
| led | Indicates the hexadecimal value displayed on the LEDs when the Configure method executes. |
| catalog | Identifies the file name of the NLS message catalog that contains all messages pertaining to this device. This includes the device description and its attribute descriptions. All NLS messages are identified by a catalog file name, set number, and message number. |
| setno | Identifies the set number that contains all the messages for this device in the specified NLS message catalog. This includes the device description and its attribute descriptions. |
| msgno | Identifies the message number in the specified set of the NLS message catalog. The message corresponding to the message number contains the textual description of the device that appears in output of commands, such as lsdev. |
| DvDr | Identifies the base name of the device driver associated with all device instances belonging to the device type. |
| Define | Names the Define method associated with the device type. |
| Configure | Names the Configure method associated with the device type. |
| Change | Names the Change method associated with the device type. |
| Unconfigure | Names the Unconfigure method associated with the device type. |
| Undefine | Names the Undefine method associated with the device type. |
| Start | Names the Start method associated with the device type. |
| Stop | Names the Stop method associated with the device type. |
| uniquetype | A key that is referenced by the PdDvLn link in CuDv object class. The key is a concatenation of the Device Class, Device Subclass, and Device Type values with a / (slash) used as a separator. For example, for a class of `disk`, a subclass of `scsi`, and a type of `670mb`, the Unique Type is `disk/scsi/670mb`. |

## 8.4.2 Create your PdDv

A PdDv entry is defined in an ASCII text file in a stanza format. To create your new PdDv entry for your new device type, it is recommended you start with the existing "other" device type PdDv for Fibre Channel or iSCSI, as shown in the example above.  When creating your PdDv entry, you will need to modify the following 5 fields in the PdDv entry to reflect your new device type.  All other fields should remain the same.

- type
- setno
- msgno
- catalog
- uniquetype

So, to create your PdDv do the following.

Your PdDv entry needs to be created in a text file with a name ending in .odmadd.  This is required for package install to work correctly. Package install looks for a file with this suffix.

Depending on if you are adding a Fibre Channel or iSCSI device, extract the "Other" PdDv entry, redirecting the output to a file with a filename ending in .odmadd:

```
odmget -q uniquetype=disk/fcp/osdisk PdDv > filename.odmadd

 or

odmget -q uniquetype=disk/iscsi/osdisk PdDv > filename.odmadd
```

Then update the type, setno, msgno, catalog, and uniquetype fields as described above. The setno, msgno, and catalog fields refer to an NLS message file that is described in a later section.

The unqiqueype is just a concatenation of the new type you chose and the existing device class and subclass.

Continue with creating your PdAt entries below.

## 8.5 PdAt (Predefined Attribute)

A PdAt entry in ODM describes a single attribute associated with a device instance, such as max_transfer. The description of most attributes contains a default value, a possible range of values, flags to indicate the type of attribute it is and whether it is user changeable, and other fields.  If a user sets a value for an attribute, that value is stored in the customized database CuAt.   The attributes are read from PdAt and CuAt by the SCSI disk configuration method.  So there is a fixed set of possible attributes defined by that method which will be loaded for a disk device.   Some of those attributes are required, others are optional.

The PdAt entries are used in several different ways.

1.  To provide information to the user.  Some of the attribute values are set by the configuration methods so that users can easily retrieve information about the hdisks that are defined on the system.  The PdAt entries serve as place holders for these values.  Examples include the attributes that contain the PVID of the hdisk or the LUN ID of the volume on the storage device. These attributes are typically displayed by the lsattr command but are not changeable by the user.

2. To allow the user to provide instructions on how the hdisk will be used.    Some attributes may be used to

inform the SCSI disk driver of parameters and uses of the hdisk. The PdAt entries for these attributes usually contain an indication of the valid values for the attribute as well as a reasonable default value. Examples include the attributes for the disk command queue depth and reservation policy. These attributes are displayed by the lsattr command and are changeable by the user.

3. To provide information about the specific device that the hdisk represents. Some attributes contain data that is required by the AIX storage stack to recognize and configure the devices that are discovered in the SAN. Some of these attributes may use some fields, such as the "deflt" or "values" fields in slightly unconventional ways. Examples of these attributes include the PdAt entries for SCSI Mode Data and the Model Map attributes that allow AIX to match the discovered device to ODM. These attributes are typically not displayed by the lsattr command and are typically not changeable by the user.

### 8.5.1 PdAt Format and Example

An example PdAt entry is shown below, followed by a description of each field. This example is the PdAt entry for the required max_transfer attribute for a Fibre Channel device. A PdAt entry is defined using an ASCII text file in stanza format. The attributes may be in the same file as the PdDv entry with which the attributes are associated.

Unused or NULL string fields are set to "" (two double-quotation marks with no separating space) and unused integer fields are set to 0 (zero).

```
PdAt:
 uniquetype = disk/fcp/osdisk
 attribute = max_transfer
 deflt = 0x40000
 values = "0x20000,0x40000,0x80000,0x100000,0x200000,0x400000,0x800000,0x1000000"
 width = ""
 type = R
 generic = DU
 rep = nl
 nls_index = 88
```

A brief description of the PdAt fields is provided below. A full description of the fields can be found in the AIX publications.

| | |
|---|---|
| uniquetype | Identifies the class-subclass-type name of the device with which this attribute is associated. This descriptor is the same as the Unique Type descriptor in the PdDv object class. It is also referenced by the PdDvLn link in CuDv object class. The uniquetype is a concatenation of the Device Class, Device Subclass, and Device Type values with a / (slash) used as a separator. For example, for a class of `disk`, a subclass of `scsi`, and a type of `670mb`, the Unique Type is `disk/scsi/670mb`. |
| attribute | Identifies the name of the device attribute. |
| deflt | Default value. If there are several choices or even if there is only one choice for the attribute value, the default is the value to which the attribute is normally set. |

| | |
|---|---|
| values | Identifies the possible values that can be associated with the attribute name. The format of the value is determined by the Attribute Representation Flags (rep field below). For regular attributes, the possible values can be represented as a string, hexadecimal, octal, or decimal. In addition, they can be represented as either a range or an enumerated list. |
| width | This field is used if the attribute type field is M, O, or W. For all other attributes, a null string is used to fill in this field. This field is generally not used for disk attributes. |
| type | Attribute type. Identifies the attribute type. Only one attribute type must be specified. Disk attributes are generally "regular" attributes (type R). A description of the other attribute types may be found in the AIX publications, under the PdAt class. |
| generic | Generic Attribute Flags. Identifies the flags that can apply to any regular attribute, such as if the attribute is displayed by the lsattr command and if the attribute is user changeable |
| rep | Attribute Representation Flags. Indicates the representation of the regular attribute values, such as numeric or string. For group and shared attributes, which have no associated attribute representation, this descriptor is set to a null string. |
| nls_index | Identifies the message number in the NLS message catalog of the message containing a textual description of the attribute. Only displayable attributes, as identified by the Generic Attribute Flags descriptor, need an NLS message. If the attribute is not displayable, the NLS index can be set to a value of 0. The catalog file name and the set number associated with the message number are stored in the PdDv object class. |

#### 8.5.1.1 Concurrently Changeable Attributes

The "generic" field of the PdAt stanza contains flags that indicate if an attribute is displayed by the lsattr command (D), if the attribute may be updated by the user (U), and if the update can be done concurrently(C). For changeable attributes, the chdev command is used to set new attribute values. Normally the chdev command can only be executed when the disk being changed is not in use (not open) because the chdev command performs an rmdev and mkdev operation to make the attribute change take effect.

Some attributes support being changed "concurrently", while the disk is open. These changes take effect immediately even if the disk is currently open and in use. The "C" flag is used to identify such attributes. So any attribute that supports concurrent update should have generic="DUC" in the PdAt stanza.

Concurrent update support also requires support within the disk change methods and the device driver or PCM. So, only certain disk and PCM attributes allow concurrent update, and only on particular technology levels or service packs of AIX that include the support. It is recommended that these attributes use generic="DUC" as there is no negative effect of specifying that concurrent update is allowed on the attribute on code levels that do not support concurrent update.

The disk attributes that allow concurrent update are:

**location**
**reserve_policy**
**PR_key_value**
**queue_depth**
**max_retries**

The PCM attributes that allow concurrent update are:

**hcheck_cmd**
**hcheck_interval**
**hcheck_mode**
**algorithm**
**timeout_policy**

## 8.5.2 Disk Attributes

The following sections describe all of the disk attributes for non-MPIO disks. The section title is the attribute name. Each section indicates if the attribute is required or optional for Fibre Channel and iSCSI, gives the text description used by AIX, and gives an example of the values and default value for the attributes, the suggested contents of the "generic" field in PdAt and then describes the attribute in more detail.

The text description is what AIX currently uses in its message catalog, and shows up in the output of the lsattr command. It is recommended that vendors use the same text in their catalog to avoid confusion. These text messages must appear in the same catalog that has the description of the disk device, so they must be repeated in the vendor supplied catalog.

The values and defaults shown are for current versions of AIX. Generally, for optional attributes, if the attribute is not supplied for the disk, it will take the default value specified here.

### 8.5.2.1 pvid

Required for FC and iSCSI
Text Description: Physical Volume Identifier
Values: ""
Default "none"
Generic: "D" (Displayable)

This attribute is used to hold the physical volume identifier for LVM once the device is added in as part of a volume group. This attribute is set by AIX. This attribute is only for disks. It is not used for tapes.

### 8.5.2.2 pv

Optional for FC and iSCSI
Text Description: N/A

Values: "no,yes,clear"
Default "no"
Generic: "U" (User Changeable)

This attribute allows a user to manually manipulate the PVID on a disk. By including this attribute, the user can do commands like "chdev -l hdisk2 -a pv=clear" to clear a PVID from a disk or "chdev -l hdisk2 -a pv=yes" to set a PVID on a disk.

#### 8.5.2.3 scsi_id

Required for FC
Text Description: SCSI ID
Values: ""
Default ""
Generic: "D" (Displayable)

The SCSI ID of the device. This attribute is set by AIX.

#### 8.5.2.4 lun_id

Required for FC and iSCSI
Text Description: Logical Unit Number ID
Values: ""
Default ""
Generic: "D" (Displayable)

Lun of device. Set by AIX configuration

#### 8.5.2.5 ww_name

Required for FC
Text Description: FC World Wide Name
Values: ""
Default ""
Generic: "D" (Displayable)

The IEEE world wide port name of the device. This attribute is set by AIX configuration.

#### 8.5.2.6 node_name

Required for FC
Text Description: FC Node Name
Values: ""
Default ""

Generic: "D" (Displayable)

The IEEE world wide node name of the device. This attribute is set by AIX configuration.

#### 8.5.2.7 ses_attach

Required for FC and iSCSI
Text Description: N/A
Values: "no,yes"
Default "no"
Generic: ""

This attribute defines whether the device supports the SCSI ENCLOSURE SERVICES as defined in the SCSI standards. It indicates if the device is in a SCSI enclosure.  This attribute is set by AIX configuration.

#### 8.5.2.8 cfgmgr_psafe

Required for FC and iSCSI
Text Description: N/A
Values: ""
Default ""
Generic: ""

The presence of this attribute indicates to the parent configuration methods that this device's configuration method supports configuring multiple devices in parallel. Most devices should have this attribute included in its set of predefined PdAt stanzas. The known exception is if the device is managed by a non-IBM multipath product. These devices may support this attribute but must be verified with rigorous configuration and boot testing. All non-multipath and AIX MPIO devices should include this attribute.

#### 8.5.2.9 target_name

Required for iSCSI
Text Description: Target NAME
Values: ""
Default ""
Generic: "D" (Displayable)

iSCSI Qualified  Name (IQN) for the iSCSI target device.   This attribute is set by AIX configuration.

#### 8.5.2.10 host_addr

Required for iSCSI
Text Description: Hostname or IP Address
Values: ""
Default ""
Generic: "D" (Displayable)

The IP address of the iSCSI target.   This attribute is set by AIX configuration.

### 8.5.2.11 port_num

Required for iSCSI
Text Description: PORT Number
Values: ""
Default ""
Generic: "D" (Displayable)

TCP/IP Port Number used for iSCSI connections to the target.   This attribute is set by AIX configuration.

### 8.5.2.12 lun_reset_spt

Required for iSCSI, Optional for FC
Text Description: LUN Reset Supported
Values: "yes" or "yes,no"
Default "yes"
Value when option attribute no present: “no”
Generic: "DU" (Displayable)

Indicates if the device supports lun-level reset.

For iSCSI devices this should always be set to"yes", lun_level reset supported, and the only allowable value in the values field should be "yes".   For Fibre Channel, setting lun_reset_spt to “yes” is highly recommended for any multiple LUN target, though a value of "no" is allowed.  If this is set to "no", then AIX may use a target reset in cases where it needs to reset the LUN, and that may be very disruptive for other LUNs on the same target.   Note that if this attribute is not present, the assumed value is “no”, meaning that target reset is used.  Thus it is recommended that all ODM contains this attribute.

This attribute may be set as Displayable and User Changeable for FC, though it should not be changeable for iSCSI.  Even for FC, it may be wise to have the attribute set to yes, not changeable, and not displayable.

### 8.5.2.13 model_map

Required for FC and iSCSI
Text Description: N/A
Values: ""
Default See below
Generic: "" (not displayable or changeable.

The **model_map** attribute is a special attribute that does not follow the standard practice of having a set of possible values and a default value.  Instead, the "deflt" field of the ODM stanza has special meaning.

This attribute is used to match the inquiry data returned during device configuration. The deflt field consists of an offset to the inquiry data, the data length, and the actual data (represented as a string).  You may choose to match any portion of the standard inquiry data. The field typically contains pointers to the vendor and product identifier fields of the standard inquiry page. During configuration, AIX reads the inquiry data from the device and attempts to match the data read to all of the model_map stanzas in ODM. When it finds one that matches, it then uses the uniquetype of the matching stanza to define the disk at that location.

You may have multiple PdAt model_map entries for the same uniquetype, or multiple deflt values.

This attribute replaces the obsoleted model_name attribute.

The format of the "deflt" field is described in more detail below. The string may take one of several different formats. In all cases, the entire string should be enclosed in quotes.

**Format 1: SSTTstring1**

SS is inquiry data offset for string1. The offset is a 2 digit hex number.
TT is the inquiry data length for string1. The length is a 2 digit hex number.

string1 must match the standard inquiry data at offset SS for a length of TT bytes in order to match this stanza.

**Format 2: SSTTstring1/string2**

SS is inquiry data offset for string1 and string2. The offset is a 2 digit hex number.
TT is the inquiry data length for string1 and string2. The length is a 2 digit hex number.

The standard inquiry data at offset SS for TT bytes must match either string1 or string2 in order to match this stanza. The / character may be thought of as an "OR" operator.

**Format 3: Any combination of the first two formats**

The first two formats may be combined, separated by a comma. The comma character can be thought of as an "AND" operator.

SSTTstring1,UUVVstring2

SS is inquiry data offset for string1. The offset is a 2 digit hex number.
TT is the inquiry data length for string1. The length is a 2 digit hex number.
UU is inquiry data offset for string2. The offset is a 2 digit hex number.
VV is the inquiry data length for string2. The length is a 2 digit hex number.

For this stanza to be considered a match, string1 must match the standard inquiry data at offset SS for TT bytes and string2 must match the standard inquiry data at offset UU for VV bytes.

**Examples**

Format 1:

"080DMYDISK MODEL1"

This indicates the inquiry data starts at offset 8 and has a length of 0x0D (13) bytes. The inquiry data which must be matched is "MYDISK MODEL1".

Format 2:

"080DMYDISK MODEL1/MYDISK MODEL2"

This indicates the inquiry data starts at offset 8 and has a length of 0x0D (13) bytes. The inquiry data must match either "MYDISK MODEL1", or "MYDISK MODEL2", both of which are 0x0D bytes long.

Format 3:

"080DMYDISK MODEL1, 3007MYDISK2"

This indicates the device must match both of the described inquiry data strings. The first inquiry data starts at offset 8 and has a length of 0x0D (13) bytes. The inquiry data is "MYDISK MODEL1". The second inquiry data starts at offset 0x30 (48) bytes and has a length of 7 bytes. The inquiry data is "MYDISK2". Thus the device's inquiry data must match both "MYDISK MODEL1" AND "MYDISK2".

### 8.5.2.14 mode_data

Required for FC and iSCSI
Text Description: N/A
Values: See description below
Default ""
Generic: "" (not displayable or changeable)

This attribute has only a predefined default value that indicates what SCSI Mode Data settings to use for this device. This attribute works in conjunction with the "mode_default" attribute described in the next section.

The default value of the mode_data attribute indicates which bits of SCSI Mode Data the SCSI driver should attempt to set when initializing the device. The format of the mode data attribute is similar to a concatenation of the SCSI mode pages that have bits that need to be set.

The mode_data attribute default value is a string of hex digits.

The mode_data default value normally starts with the string 0x000000080000000000000200. This represents the mode parameter header (for a MODE SELECT(6) command) and the block descriptor, where the 0200 at the end is the block size of 512 bytes.

Following that prefix are a series of page descriptors. Each page descriptor starts with a 2 hex digit page number followed by a 2 hex digit page length, followed by hex digits for the number of bytes specified by the length. These bytes indicate the actual mode data that the SCSI driver should attempt to set for the device. Note that the page length does not need to match the actual length of the page, but just indicates the length of the bytes supplied in the mode_data attribute.

That sequence of page code, length, bytes may be repeated as necessary to define all required mode pages.

Note that AIX does not currently support subpages in mode data. If a device includes subpages, AIX will correctly parse the mode data, but the mode_data parameter does not allow specifying mode data values in subpages.

Here is an example of a possible default value for mode_data:

0x000000080000000000000200010124020a00000000000000000000

The first 12 bytes, through the 0200, are the header described above.

The next three bytes (010124) indicate that there is one byte of mode data for page 01, and that byte value is 0x24.

The remaining bytes (020a00000000000000000000) indicate that mode page 02 has 0x0A bytes of data, and all of those bytes should be set to zero.

### 8.5.2.15 mode_default

Optional for FC and iSCSI
Text Description: N/A
Values: See description below
Default ""
Generic: "" (not displayable or changeable)

Like mode_data, this attribute has only a "deflt" value to instruct the SCSI driver how to behave. In the case of the mode_default data, the deflt value instructs the SCSI driver which bits of mode data should be left at the drive's current value. A value of 1 for a given bit indicate to the driver to leave that bit at its current value while a value of 0 indicates to the driver that it may change that bit if there is a value in the mode_data attribute.

The format of the mode_default data starts with 0x00000000 representing the data header. The remainder of the mode_default data is the same format as the mode_data attribute. In other words, it includes strings of hex digits that represent the page code, page length, and bytes in the page, with multiple pages concatenated together. For the bytes in the page, and "1" bit value indicate bits that should be left at their current value and a "0" indicates bits that may be changed by the SCSI disk driver.

For example the mode_default attribute for osdisk is:

0x00000000010700ffff000000ff0202ffff070a00ffff0000000000ffff

This indicates mode page 01 has a length of 07, page 02 has a length of 02, and page 07 has a length of 0a. A "f" for a nibble indicates to keep the whole nibble at the drive's current value. A 0 indicates that the driver can change it if it wants.

### 8.5.2.16 location

Optional for FC and iSCSI

Text Description: Location Label
Values: ""
Default ""
Generic: "DUC" (displayable and concurrently changeable)

The location attribute is not used by the SCSI disk configuration.  It is present just to provide a place holder for the user to provide a description of the disk if the user wishes to.

### 8.5.2.17 vpd_map

Optional for FC and iSCSI
Text Description: NA
Values: See description below.
Default ""
Generic: "" (not displayable or changeable)

This attribute is used to collect device information that is displayed by the lscfg -vl command.  This is a non-displayable attribute that provides information to the SCSI driver configuration methods to inform them where to obtain various parts of the device Vital Product Data (VPD).

Note that this stanza must have the type field set to "V" (other SCSI attributes are "regular" attributes with the type set to "R").

The format for this attribute consists of one or more VPD key specifiers separated by commas.   Each key specifier has the format of either KKOOLLF or KKOOLLPPF where:

**KK** is a two letter key that represents the type of information.  Possible values and their meanings are listed below.

**OO** is the offset into the page to read the information. This value is hex formated ( 0x00-0xFF ).

**LL** is the length in bytes of data to read at the OO offset. This value is hex formated ( 0x00-0xFF ).

**PP** is the page number to access. If no page number is specified the standard inquiry page is used. This value is hex formated (0x00-oxFF ).

**F** is the format to display the data that is located at the OOLLPP.  If this value is "C" then the data is displayed in character format. If the value is a X then the data is displayed in hex format.

The accepted VPD keys (the **KK** field) are:
MF - Manufacturer
TM - Machine Type and Model
SN - Serial Number
FN - FRU Number
RL - ROS Level and ID
EC - EC Level
PN - Part Number

Z0-Z9 - Device Specific information

Note: only MF,TM and SN are required fields

For example:

"MF040AC,TM0E0DC,SN0608c2C"

This indicates:
The Manufacturer (MF) data can be found in the standard inquiry page at offset 4 and has a length of 0x0A (10) bytes. The data should be displayed as characters.

The Machine Type and Model data can be found in the standard inquiry page at offset 0E (14) and has a length of 0x0D (13) bytes. The data should be displayed as characters.

The Serial Number (SN) data can be found in mode page c2 at offset 6 and has a length of 8 bytes. The data should be displayed at characters.

#### 8.5.2.18 maxlun

Optional for FC and iSCSI
Text Description: NA
Values: 0,1
Default 1
Generic: "" (not displayable or changeable)

This attribute is generally not used for FC or iSCSI devices. If it is not present, the default value of 1 indicates that the configuration method will probe all LUNs available on the target. If this attribute is set to 0, then only LUN 0 is probed.

#### 8.5.2.19 queue_depth

Optional for FC and iSCSI
Text Description: Queue DEPTH
Values: Numeric Range
Default Single Number
Value when optional attribute not present: 1
Generic: "DU" (displayable or changeable)

This attribute indicates the "queue depth" of the device, which means the number of commands that can be queued to the device concurrently. The range of possible values and the default value are device specific. If this attribute is not included, the default value of 1 will be used. A typical default queue depth may be in the range of 16 to 32, but could be higher or lower depending on the capabilities of the device.

Note that AIX has other requirements on a device in order to support queueing of multiple commands to the device. These other requirements are:

1. The ANSI version must be 3 or larger. ANSI version is found in byte 2 of Standard Inquiry data.
2. The NormACA bit must be 1. The NormACA bit is found in bit 5 of byte 3 of Standard Inquiry data. This

bit indicates that the device supports receiving commands with the NACA bit set to 1.
3. The DQue bit must be 0. The DQue bit is in the Control Mode Page, page 0x0A, bit 0 of byte 3. If this bit is changeable, the mode data ODM must be included to ensure DQue is set to 0.

If the queue_depth attribute is not present, the device driver will not queue multiple commands to the device.

### 8.5.2.20 reassign_to

Optional for FC and iSCSI
Text Description: REASSIGN time out value
Values: Numeric Range
Default 120
Generic: "DU" (displayable or changeable)

This is a timeout value specific to the SCSI REASSIGN command. This attribute indicates how many seconds the SCSI driver waits for a REASSIGN to complete. The default value is 120 seconds.

### 8.5.2.21 start_to

Optional for FC and iSCSI
Text Description: START unit time out value
Values: Numeric Range
Default: 60
Generic: "DU" (displayable or changeable)

This is a timeout value specific to the SCSI START UNIT command. This attribute indicates how many seconds the SCSI driver waits for a START UNIT to complete. The default value is 60 seconds.

### 8.5.2.22 rw_timeout

Optional for FC and iSCSI
Text Description: READ/WRITE time out value
Values: Numeric Range
Default: 30
Value when optional attribute not present: 30
Generic: "DU" (displayable or changeable)

This is a timeout value specific to the SCSI READ or WRITE commands. This attribute indicates how many seconds the SCSI driver waits for READ or WRITE commands to complete. The default value is 30 seconds. AIX has also used 30 seconds as the minimum value for rw_timeout for FC attached disks. However there is no particular limitation within the FC drivers that require a time out value that large. So starting with AIX 7.2TL5, AIX will allow setting rw_timeout as low as 10 seconds for some IBM devices. Storage vendors are encouraged to consider whether or not their device can safely use a timeout value shorter than 30 seconds, perform any required validation tests of the shorter timeout, and update their ODM packages if appropriate.

### 8.5.2.23 safe_relocate

Optional for FC and iSCSI

Text Description: N/A
Values: 0,1
Default: 1
Generic: "" (not displayable or changeable)

Indicates if the device supports block reassigment, the REASSIGN command, and wants AIX to use that command for bad block relocation. If this attribute is not present, the default value is 1. For RAID storage devices, this attribute should be included with a default value in ODM of 0, and 0 as the only acceptable value in the "values" field. This attribute is only for disks. It is not used for tapes.

#### 8.5.2.24 extended_rw

Optional for FC and iSCSI
Text Description: N/A
Values: 0,1
Default: 1
Generic: "" (not displayable or changeable)

This attribute indicates if a device supports "extended" (10 byte) READ and WRITE commands. If not present, the default value for this attribute is 1 indicating such commands are supported.

#### 8.5.2.25 rw16

Optional for FC and iSCSI
Text Description: N/A
Values: 0,1
Default: 0
Generic: "" (not displayable or changeable)

This attribute indicates if a device supports 16 byte READ and WRITE commands. If not present, the default value for this attribute is 0 indicating such commands are not supported.

#### 8.5.2.26 reset_delay

Optional for FC and iSCSI
Text Description: N/A
Values: 0,2
Default: 0
Generic: "U" (not displayable but changeable)

Indicates if the device requires the initiator to delay after sending a target or LUN reset before sending new commands. Only 2 values are supported 0 (no delay) or 2 (7 second delay). Defaults to 0, no delay.

#### 8.5.2.27 recovery_limit

Optional for FC and iSCSI
Text Description: N/A
Values: 1-4
Default: 1
Generic: "" (not displayable or changeable)

Maximum allowable recovery level before a device recovered error is reported to be re assigned.
Defaults to 1.

### 8.5.2.28 q_type

Optional for FC and iSCSI
Text Description: Queuing TYPE
Values: "none,ordered,simple"
Default: "simple"
Generic: "DU" (displayable and changeable)

Indicates the device's queue type, the way commands are queued to the device. The values are "ordered", "simple", or "none". If this PdAt does not exist, it defaults to "none", commands not queued to the device. This attribute is only for disks. It is not used for tapes. Most devices should be set to "simple".

### 8.5.2.29 ucfg_stopu

Optional for FC and iSCSI
Text Description: NA
Values: "no,yes"
Default: "yes"
Generic: "DU" (displayable and changeable)

Indicates if a STOP UNIT should be issued when a device is unconfigured. It indicates if the device should be spun down on unconfigure. Defaults to "yes". For RAID storage devices, it is likely that this could be set to "no" as the default value and only possible value.

### 8.5.2.30 q_err

Optional for FC and iSCSI
Text Description: Use QERR bit
Values: "no,yes"
Default: "yes"
Generic: "DU" (displayable and changeable)

Setting of low order QErr bit on mode page 0xA. This attribute indicates whether the low order QErr bit should be turned on in the mode data, control mode page. The low order bit must be changeable (as indicated by the changeable parameters returned by mode sense) to use this attribute. If the bit is changeable, this attribute also indicates to AIX how to handle the pending queue on check condition.

If the low order QErr bit is not changeable, AIX uses the clr_q attribute to determine what to do with pending commands after check condition.

The attribute's values of "yes" and "no" map to the following QErr settings:
    A value of no = x0b - bit not turned on
    A value of yes = x1b - bit turned on.
        where x means don't care (can be 0 or 1)

Defaults to "yes", device driver will set QErr bit if it is supported by the device.

Note: The q_err attribute only applies to the LOW ORDER bit of the QErr field. The value of the HIGH ORDER QErr bit is specified in the mode data attribute. This attribute is only for disks. It is not used for tapes.

#### 8.5.2.31 clr_q

Optional for FC and iSCSI
Text Description: Device CLEARS its Queue on error
Values: "no,yes"
Default: "no"
Generic: "DU" (displayable and changeable)

This attribute is only used if the low order QErr bit is not changeable (see q_err descr. above). This attribute indicates if a device clears its queue on error. If the low order QErr bit is not changeable, then this attribute tells AIX what the device does with its queue of pending commands after check condition.

Defaults to "no", device does not clear its queue on error.

Note: The clr_q attribute only applies to the LOW ORDER bit of the QErr field. The value of the HIGH ORDER QErr bit is specified in the mode data attribute. This attribute is only for disks. It is not used for tapes.

#### 8.5.2.32 max_transfer

Optional for FC and iSCSI
Text Description: Maximum TRANSFER size
Values: Numeric List
Default: Number
Generic: "DU" (displayable and changeable)

Maximum transfer size in bytes that can be transferred to the device in one operation. The adapter also has a maximum transfer size. The smaller of the device and adapter transfer size will be used. Possible values are from 128K to 16 MB. The default is 256K (0x40000).

The field should be a list of possible values, like powers of 2 such as 128k, 256k, 512k, and so on.

#### 8.5.2.33 block_size

Optional for FC and iSCSI
Text Description: BLOCK size
Values: 512, 1024, 2048
Default: 512
Generic: "DU" (displayable and changeable)

The device's block size. Defaults to 512. This attribute is primarily for optical drives which may have block sizes of 1024 or 2048.

### 8.5.2.34 max_coalesce

Optional for FC and iSCSI
Text Description: Maximum Coalesce size
Values: Numeric List
Default: Number
Generic: "DU" (displayable and changeable)

Indicates maximum number of bytes that can be coalesced. If there are multiple pending contiguous SCSI operations, they may be combined into one operation. This attribute defines the largest size of one of these combined operations. This value should be the same as or smaller than max_transfer.

Defaults to 0x10000. This attribute is only for disks. It is not used for tapes.

### 8.5.2.35 reserve_policy

Optional for FC and iSCSI
Text Description: Reserve Policy
Values: no_reserve,single_path,PR_exclusive,PR_shared
Default: single_path
Value when optional attribute not present: single_path
Generic: "DUC" (displayable and concurrently changeable)

Indicates the type of SCSI reservation (if any) to be established and managed by the device driver while the disk is open in normal mode. It can be set to "no_reserve", "single_path", "PR_exclusive", or "PR_shared", which are detailed below.

no_reserve : no reservation command is sent to the device while it is open.

single_path : a SCSI Reserve command is issued to the device when it is opened normally. A SCSI Release command is issued to the device when it is closed. Only devices which support the SCSI Reserve/Release commands should provide this attribute value.

PR_exclusive : SCSI Persistent Reservation commands are issued to the device when it is opened normally. The Persistent Reservation commands are issued and managed in such a manner as to provide exclusive write access to the device from the host which has the device opened.

PR_shared : SCSI Persistent Reservation commands are issued to the device when it is opened normally. The Persistent Reservation commands are issued and managed in such a manner as to provide shared write access to the device from multiple host which have the device opened.

Defaults to "single_path".

Only devices which support the SCSI Persistent Reservation Command Set should provide the PR_shared or PR_exclusive attribute values.

When specifying a reserve_policy attribute value of either PR_shared or PR_exclusive, the AIX SCSI disk device driver will make use of the SCSI Persistent Reservation commands while managing access to the device. The specific SCSI Persistent Reservation commands AIX uses are based upon the Persistent Reserve In and Persistent Reserve Out commands as defined in the SCSI Primary Command document. Described below are what AIX requires of a target in order for it to support a reserve_policy attribute value based upon Persistent Reserve (PR_shared or PR_exclusive).

- The device must support the APTPL bit in the PR out parameter list. The standard indicates that support for this is optional, however AIX requires this bit be supported in order for the device to work with our design.

- The device must support PR Out service action codes 00 through 06 .

- The device must support the Report Capabilities Service Action, and return values that include the TMV (Type Mask Valid) bit set to 1 indicating that the PR type bit mask indicates which PR types the device supports.

- The device must support Persistent Reservation Type Codes 01, 03, 05, 06, 07, and 08.

- The device does NOT need to support Persistent Reservation Scope Codes other than zero (logical unit).

- The device ODM must include the PR_key_value attribute described below.

#### 8.5.2.36 PR_key_value

Optional for FC and iSCSI
Text Description: Persistent Reserve Key value
Values: ""
Default: "none"
Generic: "DUC" (displayable and concurrently changeable)

Indicates the value of the Persistent Reservation registration key to be used whenever reserve_policy is configured with a value of PR_shared or PR_exclusive. Thus, this attribute is only needed if the reserve_policy attribute has PR_exclusive or PR_shared as one of its possible values.

The default value for PR_key_value is "none".

Note: It is required that a valid PR_key_value be established before the PR_exclusive or PR_shared reservation policies can be used for a device. When using one of these reservation_policies for an MPIO enabled device, all initiators within the host which are attached to the device will be registered with this same registration key value.

#### 8.5.2.37 pr_options

Optional for FC and iSCSI
Text Description: N/A
Values: "0,1"
Default: "0"
Generic: "DU" (displayable and changeable)

The pr_options attribute should be included if the device supports Persistent Reserve reservation policy, and must be included if PR is supported but the "Report Capabilities" PR IN Service Action does not set the TMV bit to 1. Including this attribute reduces the need for AIX to issue the Report Capabilities PR IN command during disk configuration, which may speed up the configuration of many disks.

This attribute is set to "1" if the device supports PR Type 7 (Write Exclusive -- All Registrants) and it is set to 0 if PR Type 7 is not supported by the device.

### 8.5.2.38 unique_id

Optional for FC and iSCSI
Text Description: Unique device identifier
Values: See text below
Default: ""
Generic: "D" (displayable but not changeable)

This attribute provides a map to information used to create a unique identifier for a device. This attribute is only for disks. It is not used for tapes. A device ODM must include either this attribute or the "sn_location" attribute described next in order to support the Fibre Channel "Dynamic Tracking" capability (i.e. setting the dyntrk FC protocol driver attribute to "yes").

The unique_id PdAt "values" field consists of one to three "sets" of values, comma delimited. Each "set" is made up of two or three hex two digit values which represent the inquiry page, offset into a page and the length of data to read. The unique_id "sets" will be used to create a unique device serial number. Each of these "sets" defines a substring of the device serial number.

The format of each set of data is OOLLPP or OOLL, where:

OO - the offset in bytes within an inquiry page
LL - the length in bytes of the value to be read
PP - the inquiry page number

The first set of data must be present and must point to a device serial number string. The second and third sets of data are optional, and point to a product identifier string and vendor identifier string, respectively. If the second or third sets are not present in the "values" field, then the product identifier and vendor identifier are obtained from the standard inquiry data fields that are used for product and vendor identification.

The appendix contains more details about the AIX unique ID attribute and unique_id generation.

### 8.5.2.39 sn_location

Optional for FC and iSCSI
Text Description: N/A
Values: See text below
Default: ""
Generic: "" (not displayable and not changeable)

The sn_location attribute provides a location for the device serial number. This attribute is required in order to support dynamic tracking if the unique_id attribute is not present. Generally, this attribute has been superseded by the unqiue_id attribute, though it is still supported.

The "values" field for the sn_location attribute is similar to the unique_id attribute's values field. In other words, the sn_location value's field consists of one or more sets of data, where each set consists of either two or three pairs of hex digits, such as OOLLPP or OOLL, where:

OO - the offset in bytes within an inquiry page
LL - the length in bytes of the value to be read
PP - the inquiry page number

These sets of data are separated  by commas.  When combined, the data from the set should create a serial number that uniquely identifies the devices.  Unlike unique_id, the product ID and vendor ID from standard inquiry data are not automatically added to the serial number string.

For example, if an inquiry to VPD Inquiry Page 0x83 is sufficient to determine a unique serial number for a device, then the **sn_location** attribute could look something like:

```
PdAt:
        uniquetype = some_class/fcp/some_type
        attribute = sn_locationocation
        deflt = "040283"
        values = ""
        width = ""
        type = R
        generic =
        rep = s
        nls_index = TBD by device owner
```

This indicates that the VPD Inquiry Page 0x83 should be retrieved.  Once retrieved, the driver should start at offset 0x04 and read 0x02 bytes worth of data to create a unique serial number for this particular LUN.

Another example would be a device that requires multiple inquiries to build a unique identifier for a given LUN:

```
PdAt:
        uniquetype = some_class/fcp/some_type
        attribute = sn_location
        deflt = ""
        values = "3604,040283"
        width = ""
        type = R
        generic =
        rep = s
        nls_index = TBD by device owner
```

In this example, two inquiries are required to build a unique serial number for the device.  The first part of the serial number is retrieved from the standard Inquiry page, offset 0x36 into this page. 0x04 bytes are read from this offset.  This 0x04 byte value is then concatenated with the 0x02 bytes of offset 0x04 of VPD Inquiry Page 0x83 to form a 6 byte unique serial number.

The maximum length of one of these comma-delineated strings is 255 bytes.

#### 8.5.2.40 scbsy_dly

Optional for FC and iSCSI
Text Description: N/A
Values: Number list or range
Default:  2
Generic: "" (not displayable and not changeable)

Specifies the number of seconds the SCSI driver should pause after receiving a SCSI Busy status.  The default value of 2 seconds will be used if this attribute is not present.  The default value is likely appropriate for all devices, as the SCSI disk driver performs more retries for SCSI busy status than for other error types.

#### 8.5.2.41 qfull_dly

Optional for FC and iSCSI
Text Description: N/A
Values: Number list or range
Default:  2
Generic: "" (not displayable and not changeable)

Specifies the number of seconds the SCSI driver should pause after receiving a SCSI Queue Full status.   The default value of 2 seconds will be used if this attribute is not present  The default value is likely appropriate for all devices.  This attribute is only for disks. It is not used for tapes.

#### 8.5.2.42 enable_rw16

Optional for FC and iSCSI
Text Description: Queue Full Delay
Values: "yes,no"
Default:  "yes"
Generic: "" (not displayable and not changeable)

Allows the SCSI driver to attempt to use a 16-byte Read Capacity command if the 10-byte Read Capacity command returns 0xFFFFFFFF as the last LBA on the device.   If this attribute is not present, the default value will be “yes”.   If “rw16” is “yes”, the driver will immediately use the 16-byte Read Capacity command and “enable_rw16” will not be used.  If “rw16” is “no”, then a 10-byte Read Capacity will be used initially, and a 16-byte Read Capacity will be sent only if the 10-byte command returns 0xFFFFFFFF and the value of “enable_rw16” is not explicitly set to “no”.

#### 8.5.2.43 lbp_enabled

Optional for FC and iSCSI
Text Description: Logical Block Provisioning Enabled
Values: "true,false"
Default:  "true"
Generic: "" (not displayable and not changeable)

Indicates that the underlying storage controller supports Logical Block Provisioning (aka Thin-Provisioning) related commands and SCSI driver can issue SCSI commands to release data blocks back to the storage controller when requested to do so.

This attribute can be safely added for any AIX version. That is, on an AIX version that does not support Thin-Provisioning yet, this attribute will be ignored.

### 8.5.2.44 LUR_on_error

Optional for FC and iSCSI
Text Description: LUN Reset required after adapter error
Values: "yes,no"
Default:  "yes"
Generic: "U" (not displayable and changeable)

Indicates that the underlying storage controller requires a LUN reset after certain adapter errors.  In some cases, when AIX issues a SCSI write command which fails with an adapter error, it may not be known if that write command and all of its data has been transferred to the device.   If the data has been transferred, then that write may complete at some time in the future, unknown to AIX since the command has already been failed within AIX by the adapter error.  This can happen if, for example, the FC fiber is pulled after the data has been transferred but before the SCSI status has returned to AIX.  If a second write occurs to the same range of LBAs that were affected by the first write, then the writes may not complete in the expected order.

If the **LUR_on_error** attribute is not present or if it is present and set to “yes”, then AIX issues a LUN Reset when there is an adapter error on a SCSI write command, to ensure that all pending commands are flushed from the device before issuing additional SCSI commands.  This guarantees that the writes complete in the expected order.

For some devices, writes to the same LBA are automatically ordered.  For example, if the device cache algorithm locks the affected LBAs, effectively serializing multiple writes to the same block, then the writes always complete in the expected order.   For such “safe” devices, setting **LUR_on_error** to “no” avoids sending the LUN Reset after an adapter error.

### 8.5.2.45 max_retries

Optional for FC and iSCSI
Text Description: Maximum number of times to try a command on a path
Values: Integer number
Default:  5
Value when attribute not present: 5
Generic: "DUC" (displayable and concurrently changeable)

The max_retries attribute controls how many times a read or write command is attempted on a path.  Note that it is slightly misnamed, as it indicates the total attempts, not the number of retries.   The AIX PCM has additional algorithms that may limit the number of retries on a path, but the AIX PCM may also try a command more than max_retries total times when considering all the paths to a disk.  When using the AIX PCM, it is recommended that this attribute is not included in the ODM, allowing the default value of 5 to be set.  If this attribute is included when the AIX PCM is in use, then the minimum value of this attribute must be set to 5.

For ODM packages that define single path (non-MPIO) disks, the minimum value for the max_retries attribute may be set to 2, though a default value of 5 is still recommended.  Lower values increase the chances that a command fails prematurely.  A small value such as 2 is intended for configurations that include a third party MPIO solution which uses a collection of single-path disks for each volume.

## 8.5.6 MPIO PdAt

This section gives an overview of additional attributes required if you chose to support MPIO (Multi-Path I/O) capable devices.

In order to make your device MPIO capable, you must provide an additional PdDv entry and several additional PdAt entries, and also some path attribute entries for PdPathAt.   MPIO devices also require a Path Control Module (PCM).   The default AIX PCM supports a specific set of devices for which AIX ships MPIO ODM.   However, the default PCM may work with other devices as long as the device does not require any SCSI or vendor specific features that the AIX PCM does not currently implement.

The default PCM handles devices that are active/active (i.e. all storage device ports are equivalent) or devices that implement ALUA (Asynchronous Logical Unit Access).   For ALUA devices, the PCM only implements "implicit ALUA" but does not implement "explicit ALUA".  For the AIX PCM, the device should not have any port groups that remain in "transitioning" state for an extended period of time.   If the device meets these requirements, then vendor ODM can point to the AIX default PCM to allow creation of an MPIO device.  But it is entirely up to the vendor to test the device with the AIX PCM, and no claims are made that any particular device that is not explicitly supported by AIX will work with the default PCM.

### 8.5.6.1 Device PdAt Entries

For MPIO devices, several of the attributes listed above as optional become required if the device is an MPIO device.  These required attributes are

unique_id
cfgmgr_psafe
reserve_policy
PR_key_value

One additional attribute is required as well.  That is the "PCM" attribute. The PCM attribute is displayable (genreric = "D"), regular attribute (type R) .  The text description of the attribute is "Path Control Module" and the deflt and values field should contain a uniquetype of the form "PCM/friend/<devicetype>".  This attribute serves as a link to a PCM device definition as well as PCM specific attributes.

### 8.5.6.2 PCM ODM Entries

The PCM requires one additional PdDv entry and several PdAt entries, and a PdPathAt entry.

The PdDv entry is similar to the device PdDv entry, except the class and subclass are "PCM" and "friend" and the type is vendor defined.  The DvDr field should be "aixdiskpcmke" and the Configure field should be "aixdiskpcmrtl".

The PdPathAt ODM data base is similar to PdAt, though it contains attributes that are specific to a path. PdPathAt has the same fields as PdAt, except PdPathAt is missing the type and width field as they are not needed for path attributes. The other fields have the same names and same meaning as the corresponding fields in PdAt.

All of the PCM specific PdAt and PdPathAt attributes use the uniquetype from the PCM PdDv entry ("PCM/friend/<devicetype>"), and all of the attributes are required. The attributes are explained in the following sections.

#### 8.5.6.3 dvc_support

Text Description: N/A
Values: device uniquetype list
Default: ""
Generic: "" (not displayable and not changeable)

This attribute ties the PCM entries back to the device entries. The "values" field must contain the uniquetype of the devices that point to this PCM uniquetype. The values may be a list of one or more unique types, separated by commas. So, for example, if the device with uniquetype of "disk/fcp/mydevice" has a PCM attribute with the value "PCM/friend/mydevice", then the PCM/friend/mydevice uniquetype must have a dvc_support attribute that contains "disk/fcp/mydevice".

#### 8.5.6.4 algorithm

Text Description: Algorithm
Values: "fail_over,round_robin,shortest_queue"
Default: "fail_over"
Generic: "DUC" (displayable and concurrently changeable)

This attribute indicates which path selection algorithm the PCM uses to select paths for this device. The fail_over algorithm uses just one path at a time, but must be used if the reserve_policy is single_path. round_robin alternates between paths equally, while shortest_queue acts like round_robin under light load but as the load becomes heavier, it selects the path with the fewest I/O outstanding. Note that shortest_queue is only valid for AIX 6.1TL9 or AIX 7.1TL3 or later. The default PCM in earlier versions of AIX does not implement shortest_queue.

#### 8.5.6.5 hcheck_mode

Text Description: Health Check Mode
Values: "enabled,failed,nonactive"
Default: "nonactive"
Generic: "DUC" (displayable and concurrently changeable)

The health_check_mode attribute indicates to the PCM how it should perform health check operations to determine if good paths have failed or if failed paths have recovered. The possible values are as follows:
**nonactive**: Send health check commands on paths that do not have any outstsanding I/O (i.e. paths that are not active at the time of the health check.
**failed**: Send health check commands only on failed paths.

**enabled**: Send health check commands on all enabled paths.
There is little reason to use anything other than "nonactive".

#### 8.5.6.6 hcheck_command

Text Description: Health Check Command
Values: "test_unit_rdy,inquiry"
Default:  "test_unit_rdy"
Generic: "DUC" (displayable and concurrently changeable)

This attribute indicates which SCSI command to use to perform a path health check.  The two choices indicate use of either the SCSI Test Unit Ready command or a SCSI Standard Inquiry command.

#### 8.5.6.7 hcheck_interval

Text Description: Health Check Interval
Values: Numeric range
Default:  60
Generic: "DUC" (displayable and concurrently changeable)

This attribute indicates the number of seconds between initiation of the health check process.   Setting this attribute to 0 turns off the health check, so 0 should be one of the possible values.   A default value of 60 is recommended, and it is rarely a good idea to use a value lower than that.  It is also unwise to use a value that is less than the rw_timeout value.

#### 8.5.6.8 dist_tw_width

Text Description: Distributed Error Sample Time
Values: 10-1000,1
Default:  50
Generic: "DU" (displayable and changeable)

This is a sample interval for a seldom used feature of the PCM known as "distributed error checking".  The attribute is required but there is little reason to change it.

#### 8.5.6.9 dist_tw_pcnt

Text Description: Distributed Error Sample Time
Values: 0-100,1
Default:  0
Generic: "DU" (displayable and changeable)

This is a percentage for a seldom used feature of the PCM known as "distributed error checking".  The attribute is required but there is little reason to change it.

#### 8.5.6.10 timeout_policy

Text Description: Timeout Policy
Values: retry_path, fail_path, disable_path
Default:  fail_path

Generic: "DUC" (displayable and concurrently changeable)

This attribute guides the PCM on what action to take when a command timeout occurs. The legacy behavior, indicated by "retry_path", was to retry the I/O on the same path, which may be a bad idea. The recommended value of "fail_path" causes the PCM to fail the path as long as there are other paths available and retry the I/O on one of the other paths. The final value "disable_path" causes the PCM to disable the path as long as others are available. This choice requires manual intervention from the user to re-enable the path, so fail_path is strongly recommended.

#### 8.5.6.11 priority

Text Description: Path Priority
Values: 1-255,1
Default: 1
Generic: "DU" (displayable and changeable)

This is a PdPathAt attribute. This attribute indicates the "priority" of the path, which tells the PCM either how frequently to use the path or in what order to use the paths. The default value should be 1, and most users likely leave this set at the default.

If the algorithm is fail_over, the priority is used to order the paths, with "1" being first. So in that case the path with the lowest priority value that is enabled and not failed will be for I/O. If the algorithm is round_robin or shortest_queue, this value is treated as a percentage. The number of I/Os sent on each path should reflect the path priority percentage. If there are two paths with 30 and 70, then 30 percent of the I/O will select the first path and 70 percent will select the second path.

#### 8.5.6.12 path_isid

Text Description: Path ISID
Values: Numeric values
Default: None
Generic: "U" (changeable)

This is a PdPathAt attribute for MPIO iSCSI devices. It is only required with the iSCSI MPIO support which is added to AIX 7.2 TL3. If it is present on older levels of AIX it will not be used, but will not interfere with anything either. It is not needed for FC devices.

This attribute is used by the iSCSI configuration methods to store the ISID value to be used for a particular path. It is not intended to be seen or modified by users, so it is not displayable. However, the attribute must be present in order to properly configure and MPIO iSCSI disks.

### 8.5.8 Obsoleted Attributes

Several attributes that were previously used by AIX are now obsolete. They are only listed here to avoid confusion if you come across them on an older system. The obsoleted attributes are:

- pm_dev_itime

- pm_dev_stime
- pm_devid
- pm_dev_att
- reserve_lock
- model_name (superseded by model_map)
- max_data_rate
- media_rate

### 8.5.10 Create your PdAt Entries

PdAt entries are defined in an ASCII text file using a stanza format, similar to the PdDv entry. To create your new PdAt entries for your new device type, it is recommended you start with the existing "other" device type PdAt entries for Fibre Channel or iSCSI, modifying each PdAt entry to reflect your new device. Then review the required and optional attributes lists, choose all required attributes not already in your file and any optional attributes needed by your device, and add their PdAt entries, modified for your device, to your text file. Review and possibly modify the following 5 fields in your PdAt entries to reflect the attributes of your new device type, unless otherwise stated in the previous attribute descriptions. All other fields should remain the same, unless specifically called out in the attribute description.

- uniquetype - Modify to use the uniquetype in your PdDv
- deflt - Set the desired default value for the attribute
- values - Set the appriorate "possible values" for the attribute
- nls_index - Set the index of the text string found in the NLS message catalog referenced in your PdDv entry which describes this attribute.
- rep - ensure that the value of the rep field is appropriate for the values field. Most likely this field will not be changed.

You will want to append your PdAt entries to the text file you already created containing your PdDv entry. This will allow you to run the odmadd command once to add all ODM entries.

Depending on if you are adding a Fibre Channel or iSCSI device, create the required entries by extracting the "Other" PdAt entries, appending the output to your filename.odmadd file that contains your PdDv entry:

```
odmget -q uniquetype=disk/fcp/osdisk PdAt >> filename.odmadd

 or

odmget -q uniquetype=disk/iscsi/osdisk PdAt >> filename.odmadd
```

Update the uniquetype, deflt, values, nls_index, and optionally the rep fields to reflect your new device as described above.

Determine if there are any additional required or optional attributes not already in your file that are needed for your new device. Add one PdAt entry for each attribute chosen to your existing *.odmadd file.

For example, the required model_map PdAt entry is not included in the system disk/fcp/osdisk attributes because that uniquetype is used as a default for any disk that does not match other ODM. So you will need to add a model_map attribute as described earlier to recognize your device and match it to your ODM.

## 8.6 PdAtXtd (Predefined Attribute Extended)

ODM PdAtXtd entries are used to display help text for their specific attribute in WebSM. They are paired with the PdAt entries that are displayable, updateable, or both. Each PdAtXtd entry is associated with the PdAt entry with the same attribute name. PdAtXtd entries are not provided for every PdAt meeting the criteria above. Help text must also be available to be displayed. In addition, non-displayable attributes (i.e with a null string in the 'generic' field of the PdAt object class) should not have a corresponding PdAtXtd entry, otherwise, it will become displayable in WebSM.

The PdAtXtd object class can also be used to override the current value or possible values of an attribute.

The current PdAtXtd entries found on your system should be used to determine which ones you need to add for your new device. This is further explained in the section describing how to create your PdAtXtd entries. The format of a PdAtXtd entry is shown below using the required Fibre Channel PdAtXtd attribute max_transfer as an example. This is followed by a brief description of each field.

A list of current AIX PdAtXtd entries is given for both Fibre Channel and iSCSI devices.

The section ends with a description of how to create your PdAtXtd entries.

### 8.6.1 PdAtXtd Format and Example

A PdAtXtd entry is defined using an ASCII file. An example PdAtXtd entry is shown below, followed by a description of all possible fields. This example is the PdAtXtd entry for the required max_transfer attribute for a Fibre Channel device device. Note all fields are not required in any given PdAtXtd entry. The example entry does not contain every possible PdAtXtd field.

Unused or NULL string fields are set to "" (two double-quotation marks with no separating space) and unused integer fields are set to (zero).

```
PdAtXtd:
 uniquetype = disk/fcp/osdisk
 attribute = max_transfer
 sequence = "100"
 classification = "B"
 help = "2080079"
```

A brief description of the PdAtXtd fields is provided below.

| | |
|---|---|
| uniquetype | Identifies the class subclass type name of the device to which this attribute is associated. |
| attribute | Identifies the device attribute. This has the same value as the attribute field of the associated PdAt. |
| sequence | Identifies the number used to position the attribute in relation to others on a panel/menu. This field is identical to the 'id_seq_num' currently in the sm_cmd_opt |

| | |
|---|---|
| | (SMIT Dialog/Selector Command Option) object class. |
| classification | Identifies the device attribute's classification. The valid values are B, A, or R, which indicate a basic attribute, advanced attribute, or required attribute, respectively. |
| operation | Identifies the type of operation associated with the unique device type. Operation and attribute name fields are mutually exclusive. |
| operation_value - | Identifies the value associated with the Operation field. |
| description | Identifies the attribute's description. |
| list_cmd | Identifies the command to issue to override the attribute's current value, except when operation field is set, then it will be the command to issue to return information associated with the operation. |
| list_values_cmd - | Identifies the command to issue in order to obtain the possible values of an attribute. The values returned will override the values field in the Predefined Attribute object class. |
| change_cmd | Commands used to change the attribute value if 'chdev' cannot be used. |
| help | Help text associated with the attribute. |
| nls_values | Identifies the text associated with the attribute's values. These values will be displayed in place of the values stored in the Predefined Attribute object class. The ordering of values should match the ordering in the Predefined Attribute values field. |

### 8.6.2 Required PdAtXtd Entries

Below is a list of the current PdAtXtd entries.  The current PdAtXtd entries found on your system should be used as the definitive guide to determine which ones you need to add.  They take precedence over the list below.  Determining the current PdAtXtd's on our system is detailed in the "Create your PdAtXtd Entries" section that follows.

Current PdAtXt entries for Fibre Channel and iSCSI:

The current list is the same for both Fibre Channel and iSCSI   (note pv, pvid, queue_depth, q_type, q_err, and clr_q are for disks only not tape devices.

- max_transfer
- assign_icon - operation field value
- pv
- pvid
- queue_depth
- q_type
- q_err
- clr_q

- rw_timeout
- start_timeout
- reassign_to

### 8.6.3 Create your PdAtXtd Entries

PdAtXtd entries are defined in an ASCII text file using a stanza format, similar to the PdDv entry. To create your new PdAtXtd entries for your new device type, it is recommended you start with the existing "other" device type PdAtXtd entries for Fibre Channel or iSCSI, modifying each PdAtXdt entry to reflect your new device.

The current PdAtXtd entries found on your system should be used to determine which ones you need to add for your new device. They take precedence over the list in the previous section. You should only add entries that already exist on your system for type "other" device, and that also have a corresponding PdAt for your device. The one exception is the "assign_icon" PdAtXtd entry will not have a corresponding PdAt entry on your system.

When creating your PdAtXtd entries, you will need to modify the following 3 fields in the PdAtXtd entry to reflect your new device type. All other fields should remain the same.

- uniquetype - Modify to use the uniquetype in your PdDv
- description - delete field if it exists in this entry
- nls_values - delete field if it exists in this entry

Note: The help text reference must remain the same, as it references an existing message file or SMIT identifier tag.

So, to create your PdAtXtd entries:

Depending on if you are adding a Fibre Channel or iSCSI device, create the required entries by extracting the "Other" PdAtXtd entries, appending the output to your filename.odmadd file that contains your PdDv entry:

```
odmget -q uniquetype=disk/fcp/osdisk PdAtXtd >> filename.odmadd

 or

odmget -q uniquetype=disk/iscsi/osdisk PdAtXtd >> filename.odmadd
```

Compare your PdAtXtd entries with your existing PdAt entries you have created for your new device. Delete any PdAtXtd entries that do not have a corresponding PdAt entry for your new device, with the exception of assign_icon. You should keep this PdAtXtd entry if you have it.
Update the uniquetype, description, and nls_values fields to reflect your new device.

- The uniquetype field should contain same value as the uniquetype field in your PdDv.
- The description field should be completely removed if it exists.
- The nls_values field should be deleted if it exists.

Your filename.odmadd file should now be complete and ready for testing. After testing, you should save it so it can be included in your install package.

## 8.7 Test filename.odmadd File

After you have added your PdDv, PdAt, and PdAtXtd entries to your *.odmadd file, you will need to test the file to ensure the syntax is correct. Do this by adding the entries to the ODM database, retrieving them, and then removing them. The ODM commands needed to do this are odmadd, odmget, and odmdelete respectively. A description of these commands can be found in the man pages or online documentation.

The odmadd command is used to add entries to the ODM database. Note that ODM allows duplicate entries, so running odmadd twice on the same file will result in duplicate entries, which may cause issues. Consequently, you will always need to delete any existing ODM entries before re-adding them to avoid having multiple copies of the same entry on the system.

To test your filename.odmadd file:

1. Add your PdDv, PdAt, and PdAtXtd entries to the ODM database:

   ```
   odmadd filename.odmadd
   ```

   This will add all entries in your *.odmadd file

2. Retrieve your ODM entries so you can verify they have been added. You can use your device's uniquetype value to query the database for objects specific to your device. If you want to only retrieve one attribute, as opposed to all of your device's attribute, you can use multiple search criteria for your query.

   For example, to retrieve all PdDv, PdAt, and PdAtXtd entries for a device with uniquetype disk/fcp/ihvdisk:

   ```
    odmget -quniquetype=disk/fcp/ihvdisk PdDv

    odmget -quniquetype=disk/fcp/ihvdisk PdAt

    odmget -quniquetype=disk/fcp/ihvdisk PdAtXtd
   ```

   To retrieve one attribute, e.g max_transfer:

   ```
   odmget -q "uniquetype=disk/fcp/ihvdisk AND attribute=max_transfer" PdAt
   ```

   To retrieve on extended attribute, e.g max_transfer:

   ```
   odmget -q "uniquetype=disk/fcp/ihvdisk AND attribute=max_transfer" PdAtXtd
   ```

3. Delete all of your ODM entries so when you install your package, you won't get multiple instances of the same entries. The odmdelete command uses the same query syntax as odmget, however

odmdelete requires that the -o flag is used to specify which database to manipulate.

For example, to delete all PdDv, PdAt, and PdAtXtd entries for a device with uniquetype disk/fcp/ihvdsk, do the following commands,

```
 odmdelete -quniquetype=disk/fcp/ihvdisk -o PdDv


 odmdelete -quniquetype=disk/fcp/ihvdisk -o PdAt


 odmdelete -quniquetype=disk/fcp/ihvdisk -o PdAtXtd
```

To delete one attribute, such as max_transfer:

```
odmdelete -q "uniquetype=disk/fcp/ihvdisk AND attribute=max_transfer" -o PdAt
```

# 9 Message Catalog

A messages catalog contains the text strings that are displayed when the ODM database is queried and the output is displayed, such as when using the lsdev or lsattr commands. It also contains text strings displayed in SMIT and WebSM.

The message catalog is created from a message source file, which is an ASCII text file with a filename ending in .msg. Messages are grouped into one of more sections of the *.msg file. These sections are referred to as sets. Each set is numbered, and each message is also prefixed with a unique number. The combination of the set number and message number are used to find the exact message in the message catalog.

The message catalog file is a non-text file that is created from the message source file, *.msg, using the gencat command. Gencat creates a message catalog file name ending in .cat. This message catalog file is included in your install package. The default English message catalog files can be found under /usr/lib/nls/msg/en_US. Message catalogs used by the AIX configuration subsystem, such as the message catalog created for storage device ODM, are installed under /usr/lib/methods.

Your new device type ODM entries will reference messages in your new message catalog file. The catalog field of your PdDv entry should be set to your new *.cat filename, the setno field of the PdDv should reflect the set number, and the msgno field should reference the message number describing your new device type (e.g Other FC SCSI Disk in example). The nls_index field in the PdAt is assigned the message number for that attribute's text description. All PdAt messages are located in the message catalog referenced in the PdDv, using the set number referenced in the PdDv.

When adding your new device, you will only need to have one message set in your message catalog. It is not necessary to have multiple sets.

Two commands are available that allow you to display the messages in a message catalog. The commands are:

| | |
|---|---|
| dspcat | Displays the messages contained in the specified message catalog. The following example displays the messages located in the x.cat message source file: `dspcat x.cat` |
| dspmsg | Displays a single message from a message catalog. The following example displays the message located in the x.cat message source file that is in message set 2 with message ID number 1. `dspmsg x.cat -s 2 1` You can use the dspmsg command in shell scripts when a message must be obtained from a message catalog. |

Note:

For Internationalization, or Multiple Language Support, you would have to provide multiple message catalog files, one for each language. This will not be described in this document. Please refer to online AIX documentation if you are interested.

## 9.1 Message Source File Syntax

This section describes the syntax of the message source file that you will need to create. In the *.msg file, comments are prefixed with a $ and one or more spaces. A message section (set) is identified by having its first line start with $set followed by a space and then the set number. Message strings start with a unique message number, followed by a single space. The message strings must be surrounded by double quotes. Some specific limitations are:

- One and only one blank character must exist between the message ID number (or identifier) and the message text.
- Message ID numbers must be assigned in ascending order within a single message set, but need not be contiguous. 0 (zero) is not a valid message ID number.
- You can include a comment anywhere in a message source file except within message text. Leave at least one space or tab (blank) after the $ (dollar sign).
- All text following the blank after the message number is included as message text, up to the end of the line. Use the escape character \ (backslash) to continue message text on the following line. The \ (backslash) must be the last character on the line that is to be continued.
- The maximum number of bytes allowed for a message is NL_TEXTMAX, which is defined in /usr/include/limits.h and is set to 8192.

The description above should be sufficient for creating your file. If you feel you need to provide something more elaborate, please refer to the "Message Facility Overview" in the online AIX publications.

## 9.2 Example Message Source File (*.msg)

Below is an example of a message source file, *.msg. Comments are prefixed with a $ and one or more spaces. A message section (set) is identified by having its first line start with $set followed by a space and then the set number. Message strings start with a unique message number, followed by a single space. The message strings must be surrounded by double quotes.

Example *.msg file:

```
$ IHV New Device
$
$set 1
1 "Other FC SCSI Disk Drive"
2 "SCSI ID"
3 "Logical Unit Number ID"
```

## 9.3 Create and Test your Message Catalog (*.cat)

The message file described above may be created in any text editor as a standard ASCII file. The message source file should contain a message to describe the new device, to be displayed by lsdev, and messages for each of the attributes included for the device, displayed by lsattr. It is recommended that the standard attribute descriptions are used to be consistent with other devices.

Your message source file must have a .msg type, such as IHV.msg, and must be converted to a message catalog file, *.cat, which your package will install on the system. The gencat command is used to create your message catalog file. The following example takes the `IHV.msg` message source file as input and generates the catalog file IHV.cat:

```
gencat IHV.cat IHV.msg
```

Your message catalog file can be named anything you want, as long as its suffix is .cat. You can look on your AIX system under /usr/lib/methods to see examples of current message catalogs shipped on your system.

Note any given file (full path included) can only be included in one fileset in AIX. So if you already have created a package containing the file /usr/lib/methods/IHV.cat, you will either need to use a different name for this package's catalog file, or update your original package's IHV.cat file and create a requisite (coreq) to the original fileset in this package.

Your install package should install your message catalog file under /usr/lib/methods.

After you have created your *.msg file, you are ready to create and test your message catalog file, as described below:

1. Create your message source file, *.msg, starting with the one provided. For this example, we will call it IHV.msg.

   Create your *.cat file using gencat:

   ```
   gencat IHV.cat IHV.msg
   ```

2. Install *.cat file on your test machine.

   ```
   cp *.cat /usr/lib/methods
   ```

3. If you have added your ODM entries for your new device (*.odmadd file) and discovered some devices of your new type, run commands to verify strings are correct:

   ```
   lsdev -Cc disk
   lsattr -El hdisk2
   ```

4. Use the dspcat and dspmsg commands to display messages in your message catalog:

   To display your entire message catalog named, for example, ihv.cat:

   ```
   dspcat ihv.cat
   ```

   To display a single message in your catalog, for example message 5 in set number 1:

   ```
   dspmsg ihv.cat -s 1 5
   ```

5. After you are satisfied with your testing, be sure to remove your message catalog file from /usr/lib/methods. This will insure when you test your install package, when you see the catalog file there, you will know your package installed it.

# 10 Install and Packaging

AIX supports two types of install package formats:

- installp
- rpm

For Aix, installp is the preferred method to create and install customer packages. It is integrated into the AIX SMIT system administration tool and customers expect AIX packages to use this format.

RPM is supported, but will not be described in this document. If you chose to use rpm as your packaging and install tool, remember to provide a post-install script that calls odmadd to add your ODM entries. You will also need to provide a pre-remove script that deletes the same entries from ODM.

To be able to create your install package, you will need to make sure you have the bos.adt.insttools package installed on your machine. To determine if you already have the package, run the following command:

lslpp -l | grep bos.adt.insttools

If you are missing the package, you need to install it before proceeding.

The Install and Packaging section is grouped into 2 large subsections.

The first subsection, "Installp Package Background and Overview", provides background on AIX installp packaging by giving an overview of the installp packages, what they are comprised of, and the installp command. If you are already familiar with AIX installp concepts, you may want to read the first 3 paragraphs and then skip the remainder of the section, returning to it for reference as needed.

The second subsection, "How to Create Your Installp Package", describes in detail what you need to do to create your package. It details the files you will need to gather and create, product coreqs, description, and versioning, and finally how to create and test your package.

This document only covers what you need to create your specific package. If you would like more detail concerning all options witih AIX packaging, please refer to the online AIX documentation for the mkinstallp command.

## 10.1 Installp Package Background and Overview

An AIX installp package consists of multiple files bundled together into a single file created in a backup file format. This single file contains the package and is sometimes referred to as the BFF (Backup File Format) image.

An AIX installp package contains one or more filesets. A fileset is simply an arbitrary grouping or collection of files. For the purposes of adding ODM entries and a message catalog for your new device, you will need only one fileset in your package. An example of when you might have additional filesets would be if you chose to support multiple languages. In that case, you would have an additional fileset for each additional language.

The easiest way to create your install package is to use the mkinstallp utility. A detailed description of how to create your package will be given in the "How to Create Your Installp Package" section. The remainder of this section gives an overview of the files in an installp package, package naming conventions, and an overview of the installp command.

### 10.1.1 Contents of an Installp Software Package

Below is a description of the files contained in an installation or update package. Each one is described in the sections below.

File path names are given for installation package types. For update packages, wherever PackageName is part of the path name, it is replaced by PackageName/FilesetName/FilesetLevel. Creation of update packages is not covered by this document.

Your package will contain the following files:

- ./lpp_name: This file provides information about the software package to be installed or updated. For performance reasons, the lpp_name file should be the first file in the backup-format file that makes up a software installation package.
- ./usr/lpp/PackageName/liblpp.a: This archive file contains control files used by the installation process for installing or updating the usr part of the software package.
- All files, backed up relative to root, that are to be restored for the installation or update of the usr part

of the software product.

### 10.1.2 File lpp_name

Each software package must contain the lpp_name package information file. This file is automatically created for you when using the mkinstallp utility. The lpp_name file gives the installp command information about the package and each fileset in the package.

### 10.1 3 File liblpp.a

The liblpp.a file will be created automatically for you when using the mkinstallp utility. The liblpp.a file is an archive file that contains the files required to control the package installation. You can create a liblpp.a file for your package using the ar command. This section lists many of the files you can put in a archive.

Throughout this section, Fileset appears in the names of the control files. Fileset represents the name of the separate fileset to be installed within the software package. For example, the apply list file is described as Fileset .al. The apply list file for the `bos.net.tcp.client` fileset in the bos.net package is `bos.net.tcp.client.al`.

For any files you include in the liblpp.a archive file other than the files listed in this section, you should use the following naming conventions:

- If the file is used in the installation of a specific fileset, the file name should begin with the Fileset. prefix.
- If the file is used as a common file for several filesets in the same package, the file name should begin with the lpp. prefix.

Below is a list of the files that are required in your package that will go in the liblpp.a archive:

- Fileset.al
- Fileset.copyright
- Fileset.inventory
- Fileset.odmadd
- Fileset.unodmadd
- Fileset.odmdel
- Fileset.pre_d
- Fileset.size

The *.al, *.size, and *.inventory files are automatically generated by the mkinstallp utility. A brief description of the files is given below. You will have to create the other files. Details of how to do this are given in the "How to Create Your Installp Package" section.

### 10.1.4 File *.al

The filesetname.al file will be automatically generated for you by the mkinstallp script. This file is referred to as the apply list. This file is a simple list of all the files in the package to be restored for this fileset. It does not include the files lpp_name or liblpp.a or the files archived in liblpp.a. Files are listed one per line with a path relative to root, as in `./usr/bin/pickle`. An apply list file is required if any files are delivered with the fileset or fileset update.

### 10.1.5 File *.inventory

The filesetname.inventory file will be automatically generated for you by the mkinstallp script. This file contains required software vital product data for the files in the fileset or fileset update. The inventory file is a stanza-format file containing an entry for each file to be installed or updated.

### 10.1.6 File *.size

The filesetname.size file will be automatically generated for you by the mkinstallp script. This file contains the space requirements for the files included in your fileset. The mkinstallp command uses your sizing input given in the mkinstallp template coupled with other sizing mkinstallp determines at runtime and creates this file, which is included in liblpp.a.

Specifically, the *filesetname*.size file contains a list of the directories touched by this fileset and the number of 512-byte blocks required in that directory for the fileset to install without space problems

### 10.1.7 Package and Fileset Names

Below is a description of package and filename conventions, device specific naming conventions, and fileset extension naming conventions that is provided as general background information. Your specific package and fileset naming requirements and recommendations are provided in the "How to Create Your Installp Package" section. Your package will be treated as a Device Driver package in the description below.

#### 10.1.7.1 Package and Fileset Naming Conventions

Below are the general conventions for creating package and fileset names. Specific requirements for packages that add new devices follows.

- A package name (PackageName) should begin with the product name. If a package has only one installable fileset, the fileset name can be the same as the PackageName. All package names must be unique.
- A fileset name has the form:

  `ProductName.PackageName.FilesetName.extension`

  where:

  - `ProductName` identifies the product or solution group.

    - `PackageName` identifies a functional group within the product.
    - `FilesetName` (optional) identifies a specific functional set of files and libraries to be installed.
    - `Extension` (optional) further describes the contents. See Fileset Extension Naming Conventions for more information.
- A fileset name contains more than one character and begins with a letter.
- All characters in the fileset name must be ASCII characters. Valid characters are upper and lower case letters, digits, underscores ( _ ), plus signs ( + ), and minus signs. The period ( . ) is used as a separator in the fileset name.
- A fileset name cannot end with a period or dot.
- The maximum length for a fileset name is 144 bytes.
- All fileset names must be unique within the package.

### 10.1.7.2 Special Naming Considerations for Device Driver

The configuration manager command (cfgmgr) automatically installs software support for detectable devices that are available on the installation media and packaged with the following naming convention:
devices.ODMDeviceSubclass.ODMDeviceClass.IHVSpecific.Extension

where:

- ODMDeviceSubclass specifies the Device Subclass portion of the ODM uniquetype field (e.g, fcp or iSCSI)
- ODMDeviceClass specifies the Device Class portion of the ODM uniquetype field (e.g disk, tape)
- IHVSpecific specifies your device model, or device type
- Extension follows the naming convention below (e.g rte for your package)

### 10.1.7.3 Fileset Extension Naming Conventions

You should use .rte for your package extension.

The following list provides some fileset extension naming conventions:

| Extension | Fileset Description |
|---|---|
| .adt | Application development toolkit |
| .com | Common code required by similar filesets |
| .compat | Compatibility code that may be removed in a future release |
| .diag | Diagnostics support |
| .fnt | Fonts |
| .help. Language | Common Desktop Environment (CDE) help files for a |

| | |
|---|---|
| | particular language |
| .loc.language | Locale |
| .msg. Language | Message files for a particular language |
| .rte | Run-time environment or minimum set for a product |
| .ucode | Microcode |

## 10.1.8 Installp Command Overview

The **installp** command has 5 major functions: apply, commit, reject, deinstall (remove) and cleanup. A base fileset is always committed to the system, which means that when `lslpp -l fileset` is executed, the fileset is shown in the COMMITTED state. This is called "auto-commit" of base levels, and it will happen automatically when then **-a** or **-ac** flags are used with **installp**. A fileset update can be applied and committed later or applied and committed all in one step. A fileset which is only applied will appear in the APPLIED state until it is committed or rejected. Only those filesets in the APPLIED state may be rejected. Once committed, any base fileset may be deinstalled which means that it is completely removed from the system and all install databases.

Detailed command syntax can be found in the installp man page, which can be found on your system.

The major actions that can be taken with the installp command are:

| | |
|---|---|
| Apply | When a fileset in a product installation package is applied, it is installed on the system and it overwrites any preexisting version of that fileset, therefore committing that version of the fileset on the system. The fileset may be removed if the user decides the fileset is no longer required.<br><br>When a fileset update is applied, the update is installed and information is saved (unless otherwise requested) so that the update can be removed later. Fileset updates that were applied can be committed or rejected later. |
| Commit | When a fileset update is committed, the information saved during the apply is removed from the system. Committing already applied software does not change the currently active version of a fileset. |
| Reject | When an update is rejected, information saved during the apply is used to change the active version of the fileset to the version previous to the rejected update. The saved information is then removed from the system. The reject operation is valid only for updates. Many of the same steps in the reject operation are performed in a cleanup operation when a fileset or fileset update fails to complete installation. |
| Remove | When a fileset is removed, the fileset and its updates are removed from the |

system independent of their state (applied, committed, or broken). The remove operation is valid only for the installation level of a fileset.

Executables provided within a product package can tailor processing for the apply, reject, and remove operations.

Reinstalling a fileset does not perform the same actions that removing and installing the same fileset do. The reinstall action (see /usr/lib/instl/rminstal) cleans up the current files from the previous or the same version, but does not run any of the unconfig or unpre* scripts. Therefore, do not assume that the unconfig script was run. The .config script should check the environment before assuming that the unconfig was completed.

## 10.2 How To Create Your Installp Package

This section describes what you need to do to create your installp package  It describes in detail the files and data you will need to gather or create, how to name your package, and finally how to create and test your package.

The easiest way to create your install package is to use the mkinstallp utility.  Step by step instructions for creating your package using mkinstallp are given at the end of this "How to Create your Installp Package" section.  First, you will need to create all of your required files, determine any prerequisites, and determine your product name and version number.

Specifically, to create your package, you will need to do the following, each of which is addressed in the sections below:

1. Determine your package and fileset names
2. Create your Required Package Files
3. Determine your corequisites
4. Determine your product version number (VRMF)
5. Determine your package sizing requirements
6. Create and Edit the mkinstallp template
7. Create your package using mkinstallp
8. Test your package

Note that any file may only occur in one installed fileset.   If you create multiple filesets for different devices, the filesets cannot have any file in common.  If there is a common file, then only one of the filesets may be installed at one time.  For example, if you create a single message catalog file that has messages for all of your devices, that message catalog cannot be included in all of the filesets.  Instead, include the catalog file in just one fileset, and then include corequisites in the other fileset to ensure that the fileset that includes the message catalog is always installed.

## 10.2.1 Determine Package and Fileset Names

You will need to provide a package name and a fileset name.  If you only have one fileset, which is the case here, the fileset name can be the same as the package name with the additional .rte suffix.  The naming conventions were described above in the packaging overview.

It is recommended that the fileset name has the following format.

devices.ODMSubclass.ODMClass. IHVmodel.rte

where:

- ODMDeviceSubclass specifies the Device Subclass portion of the ODM uniquetype field (e.g, fcp or iSCSI)
- ODMDeviceClass specifies the Device Class portion of the ODM uniquetype field (e.g disk, tape)
- IHVmodel specifies your device model, or device type. This is whatever you chose to denote your device model or device type.

If you are adding a new Fibre Channel disk device, your package and fileset names should be the following. Substitute whatever name you want to use to denote your device model or device type for IHVmodel.

package name: devices.fcp.disk.IHVmodel
fileset name: devices.fcp.disk.IHVmodel.rte

If you are adding a new Fibre Channel tape device, your package and fileset names should be the following:

package name: devices.fcp.tape.IHVmodel
fileset name: devices.fcp.tape.IHVmodel.rte

If you are adding a new iSCSI disk device, your package and fileset names should be the following. Substitute whatever name you want to use to denote your device model or device type for IHVmodel.

package name: devices.iscsi.disk.IHVmodel
fileset name: devices.iscsi.disk.IHVModel.rte

If you are adding a new iSCSI tape device, your package and fileset names should be the following:

package name: devices.iscsi.tape.IHVmodel
fileset name: devices.iscsi.tape.IHVmodel.rte

## 10.2.2 Create Required Package Files

You will need to include, at a minimum, the following files in your package:

- filesetname.odmadd - Your PdDv, PdAt, and PdAtXtd entries
- filesetname.odmdel
- filesetname .unodmadd
- filesetname.pre_d
- IHV.cat

- filesetname.copyright

Throughout this section, where filesetname appears in the name of the file to be created, it represents the name of your fileset that is in your package. For example, if your fileset name is devices.fcp.disk.IHVmodel.rte, then your *.odmadd file would be named devices.fcp.disk.IHVmodel.rte.odmadd.

mkinstallp requires the specified files to be prefixed with your fileset name so they will be included in your liblpp.a file.

Each new file will be addressed below.

#### 10.2.2.1 File *.odmadd

This is the filename.odmadd file that you have previously created. You need to rename this file to be filesetname.odmadd. When installp is run against your package, odmadd will automatically be invoked against any file with a name like filesetname*.odmadd suffix.

It is possible to use multiple odmadd files, as long as the name includes additional text between the filesetname and .odmadd suffix to distinguish the multiple files.

#### 10.2.2.2 File *.odmdel

The *.odmdel file is used to remove any *old* entries in the databases that will be replaced when the *fileset*.odmadd file is executed. The **installp** command executes all the *fileset**.odmdel scripts for a fileset when that fileset is being reinstalled, either through a migration, force install or update. The *fileset*.odmdel script contains a series of **odmdelete** commands which are used to delete the entries.

Your *.odmdel file must be named filesetname*.odmdel for mkinstallp to behave properly.

Your filesetname.odmdel file should use the odmdelete command to remove your PdDv, PdAt, and PdAtXtd entries. Your file should contain the following lines, modified to reflect your uniquetype.

```
odmdelete -o PdAt -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
odmdelete -o PdAtXtd -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
odmdelete -o PdDv -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
```

#### 10.2.2.3 File *.unodmadd

This file is used to remove any **new** entries in the databases that were added with the *fileset*.odmadd script if the fileset is deleted from the system. The **installp** command executes **odmdelete** with all files that match *fileset**.unodmadd.

Your *.unodmadd file must be named filesetname*.unodmadd for mkinstallp to behave properly.

Your filesetname.unodmadd file should also use the odmdelete command to remove your PdDv, PdAt, and PdAtXtd entries. Your file should contain the following lines, modified to reflect your uniquetype.

```
odmdelete -o PdAt -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
odmdelete -o PdAtXtd -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
odmdelete -o PdDv -q " uniquetype = 'disk/fcp/ihvdisk' " >/dev/null
```

It is likely that the *.odmdel and *.unodmadd files are identical.

### 10.2.2.4 File *.pre_d

When present the *filesetname*.pre_d script is run before the deinstallation of the fileset and is used to determine that fileset's de-installability.

You will want to verify you have no current active devices before your package is uninstalled. This is determined by checking to see if there are any CuDv device objects that use the uniquetype of your new ODM.

Your *.pre_d file must be named filesetname.pre_d for mkinstallp to behave properly.

Your filesetname.pre_d file should contain the following lines, modified to reflect your uniquetype and and to update the error message to make it appropriate for your package.:

```
uniquetypes="disk/fcp/ihvdisk"

for type in $uniquetypes
do
    rc=$(ODMDIR=/etc/objrepos odmget -q PdDvLn=$type CuDv)
    if [ -n "$rc" ]
    then
        echo "WARNING:  Unable to uninstall packge.  There are currently configured devices"
        exit 1
    fi
done
exit 0
```

### 10.2.2.5 File IHV.cat

This is your message catalog file, IHV.cat, that you should have already created. The name of the message catolog does not have any particular format other than the .cat suffix.

### 10.2.2.6 Copyright File

Your package is required to contain a copyright file. Your copyright file must be named filesetname.copyright, and must be the first file in the liblpp.a archive shipped with your package. Mkinstallp will include this file in the liblpp.a archive and insure it is the first file.

The copyright file is simply a list of all the copyrights needed for the source code sent with the package. The company copyright for the company that owns the code must be the first one listed in the copyright file, unless there are other legal considerations with the code being shipped. A suggested format is in the example below:

```
Licensed Materials - Property of Company


PROG#


(C) Copyright Company year [, year [,...]]


All rights reserved.


US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with Company Corp.
```

Format:

| | |
|---|---|
| *PROG#* | The product ID number for the product that this fileset belongs to. |
| *Company* | The registered name of the company holding the copyright. |
| *year* | The years that the copyright is registered for. This can be a list or range of years. |

## 10.2.3 Determine Corequisites

If the fileset depends on files from other packages, such as the message catalog example discussed in the introduction, then a corequisite may be used to ensure the filesets are installed together. We recommend only using corequisites, as opposed to prerequisites, in your package. The corequisite file is optional; if the fileset does not depend on any other filesets it may be omitted.

A corequisite indicates that the specified fileset must be installed at the specified fileset level or at a higher level for the current fileset to function successfully. At the end of the installation process, the installp command issues warning messages for any corequisites that are not met. A corequisite can be used to indicate requisites between filesets within the same package.

Syntax: *coreq Fileset RequiredFilesetLevel

The corequisite text is specified directly in the template file discussed later.

If you are adding a new Fibre Channel disk device, we recommend requiring a base version of the devices.fcp.disk.rte fileset (FC SCSI CD-ROM, Disk, Read/Write Optical Device Software) as your coreq, as shown below:

*coreq devices.fcp.disk.rte 6.1.0.0

This would work on any system installed with AIX 6.1 or greater (including AIX 7.1 systems)

If you are adding ODM for iSCSI disks, we recommend a corequisite for the iSCSI disk base fileset:

*coreq devices.iscsi.disk.rte 6.1.0.0

This would work on any system installed with AIX 6.1 or greater.

## 10.2.4 Determine Product Version Number/ VRMF

In AIX,  the product version number is referred to as a fileset revision level, the level, or the VRMF.   You will need to provide a VRMF for both your package and your fileset.

The syntax for the revision level (VRMF) is provided below:

Version.Release.Modification.FixLevel

where:

- Version is a numeric field of 1 to 2 digits that identifies the version number.
- Release is a numeric field of 1 to 2 digits that identifies the release number.
- Modification is a numeric field of 1 to 4 digits that identifies the modification level.
- FixLevel is a numeric field of 1 to 4 digits that identifies the fix level.

  A base fileset installation level is the full initial installation level of a fileset. This level contains all files in the fileset, as opposed to a fileset update, which may contain a subset of files from the full fileset.  This document only covers creating a base fileset package.

All filesets in a software package should have the same fileset level, though it is not required. For all new levels of a fileset, the fileset level must increase. The installp command uses the fileset level to check for a later level of the product on subsequent installations.

Fileset level precedence reads from left to right (for example, 7.1.0.0. is a newer level than 6.2.0.0).

We recommend starting with a VRMF of 1.1.0.0 or higherfor your package and fileset.   If you later modify the package, you can increase any of the VRMF fields (e.g. 1.1.1.0 or 1.2.0.0 or even 2.1.0.0).

## 10.2.5 Determine Package Sizing Requirements

Your installp package must contain information regarding how much extra filesystem space your new files consume. This information is used by the installation process to ensure that enough disk space is available for the installation to succeed. You only need to determine this for your *.cat file. The mkinstallp command automatically generates the space requirements for all of the files found in liblpp.a.

To determine the size of your *.cat file, do a du on the file.

Example:

```
du ihv.cat
```

This will generate the number of 512-byte blocks consumed by your file. If you fileset contains any other files or utilities, run "du" on those files as well and add the results.

Mkinstallp expects the sizing input to be the number of 512-byte blocks. This value is included directly in the mkinstallp template described below.

## 10.2.6 Create and Edit the Mkinstallp Template/Script

To create your installp package, you will want to use the mkinstallp utility. This utility can be script driven, and we recommend running it in this manner. We have provided an example template below that is compatible with the current versions of mkinstallp and makebff.pl. We recommend that you also refer to /usr/lpp/bos/README.MKINSTALLP on the machine being used to create the package. That file includes more description of the mkinstallp template and some more examples of the template format.

mkinstallp may also be run interactively, in which case it will create a template. If you have difficulty running mkinstallp with a template, run it interactively once and then use the automatically created template.

Edit the template file shown below, modifying filenames, package name, and coreqs, VRMF, product name, etc. to reflect your specific package information.

The text in bold needs to be updated to reflect your package. Other fields may be updated if appropriate.

Package Name: **devices.fcp.disk.IHVModel**
Package VRMF: 1.1.0.0
Update: n
Fileset
  Fileset Name: **devices.fcp.disk.IHVModel.rte**
  Fileset VRMF: 1.1.0.0
  Fileset Description: IHV Disk Support
  Copyright file path: **/tmp/IHVPackage/devices.fcp.disk.IHVModel.rte.copyright**
  USRLIBLPPFiles
  Pre-deinstall Script: **/tmp/IHVPackage/devices.fcp.disk.IHVModel.rte.pre_d**

```
  EOUSRLIBLPPFiles
  Bosboot required: y
  License agreement acceptance required: n
  Include license files in this package: n
  Requisites: *coreq devices.fcp.disk.rte 6.1.0.0
  Upsize: /usr 4;
  USRFiles
    /usr/lib/methods/IHV.cat
  EOUSRFiles
  ROOT Part: N
  ROOTFiles
  EOROOTFiles
  Relocatable: n
EOFileset
```

You need to do the following to modify the template:

1. Replace the Package Name devices.fcp.disk.IHVModel with your package name.
2. Replace the Package VRMF 1.1.0.0 with your Package VRMF.
3. Replace the Fileset Name devices.fcp.disk.IHVModel.rte with your fileset name, which should just be the package name with a .rte appended.
4. Replace the Fileset VRMF 1.1.0.0 with your Package VRMF.
5. Determine your Fileset Description and replace IHVModel Disk with your fileset description.
6. Replace the size of 4 512-byte block shown under Upsize: /usr/lib/methods with the number of 512-byte blocks for your *.cat file.
7. Replace the filename IHV.cat with your message catalog filename.

   Note: Any given file (full path included) can only be included in one fileset in AIX. So if you already have created a package containing the file /usr/lib/methods/IHV.cat, you will either need to use a different name for this package's catalog file, or update your original package's IHV.cat file and create a requisite (coreq) to the original fileset in this package.

## 10.2.7 Create Installp Format Package

Once the mkinstallp template is complete, you must collect and arrange the files for the package before running mkinstallp to create the package.

You will need to have a working, or temporary, directory set aside to place your files in, which mkinstallp will reference to create the package. This directory will be referred to as your base directory.

Below is a description of how to create and populate your base directory:

1. Create package directory structure and copy files to it.

   Create your temporary or working directory, which you will use to create your package. This is referred to as your base directory, or root build directory.

The top of the base directory is treated like it is the root directory on your system. Files that will be installed on the system should be placed in the same directory, relative to your base directory, as you want them to appear on your system.

For example, since your message catalog file IHV.cat is to be installed under /usr/lib/methods, you would place it under usr/lib/methods in your temporary working directory you are using for package creation.

Your message catalog file should be installed under /usr/lib/methods. The other files that have been created for the image are included in an archive file within the installp image.

The fileset*.odmadd, fileset*.unodmadd, and fileset*.odmdel files should be placed in a ".info" subdirectory of the base directory. The copyright and "pre_d" files may be placed in the base directory.

Below is an example listing of what your package directory should look like, using IHV and IHVModel as example filenames. In this example, the base directory is /tmp/IHVPackage:

```
# find /tmp/IHVPackage -type f -print
/tmp/IHVPackage/.info/devices.fcp.disk.IHVModel.rte.odmadd
/tmp/IHVPackage/.info/devices.fcp.disk.IHVModel.rte.odmdel
/tmp/IHVPackage/.info/devices.fcp.disk.IHVModel.rte.unodmadd
/tmp/IHVPackage/.info/devices.fcp.disk.IHVModel.template
/tmp/IHVPackage/devices.fcp.disk.IHVModel.rte.copyright
/tmp/IHVPackage/devices.fcp.disk.IHVModel.rte.pre_d
/tmp/IHVPackage/template
/tmp/IHVPackage/usr/lib/methods/IHV.cat
```

2. Create your package by running mkinstallp.

Run the mkinstallp command, specifying the template file with the -T flag:

```
mkinstallp -T /tmp/IHVPackage/template
```

WARNING: Invocation of mkinstallp will populate your temporary base directory with symbolic links to existing directories on your system, such as /etc/, /var, /opt, and /usr/local. Be very careful when cleaning up and deleting your temporary base directory after you have finished creating your pacakge so as to not accidentally delete any system files.

Your new bff (install image) that contains your package and fileset will be created under a tmp directory created in your base directory. So, in this example, your bff file would be found under /tmp/IHVPackage/tmp and would be named devices.fcp.disk.IHVModel.1.1.0.0.bff. So, you should now have a bff file that contains your package and fileset. Continuing with the same example, your BFF image, package and fileset names are:

| | |
|---|---|
| - Package name: | devices.fcp.disk.IHVModel |
| - Fileset name: | devices.fcp.disk.IHVModel.rte |
| - BFF image: | devices.fcp.disk.IHVModel.1.1.0.0.bff |

You will only have one file, your .bff file, which is your installp package. The fileset name is used when you are using the installp command to install your package. See the "Verify and Test Installp Package" section below for examples of how to install the package.

If you do not set any environment variables, mkinstallp will generate output similar to that below, which is correct:

```
# mkinstallp -T ./template
Using /tmp/IHVPackage as the base package directory.
Using /tmp/IHVPackage/.info to store package control files.
Cleaning intermediate files from /tmp/IHVPackage/.info.

Using ./template as the template file.
devices.fcp.disk.IHVModel 1.1.0.0 I
processing devices.fcp.disk.IHVModel.rte
creating ./.info/liblpp.a
creating ./tmp/devices.fcp.disk.IHVModel.1.1.0.0.bff
```

Verify your usr/lpp/PackageName/liblpp.a file that was created contains all of your files you copied into the .info directory as well as the *.al, *.inventory, and *size files the mkinstallp command created.

You can accomplish this by extracting them from liblpp.a with the ar command as shown:

```
cd /tmp/mypackage/usr/lpp/devices.fcp.disk.IHVModel
ar -xv liblpp.a
```

Finish with Verifying and Testing your installp package below.

## 10.2.8 Verify and Test Installp Package

Before shipping your installp package to customers, you will want to verify it includes your files as expected and that it installs, uninstalls, and reinstalls correctly. We recommend testing install using both SMIT and the command line installp utility.

To test your package, continuing with the same example as above:

1. Verify package contains what you think and extract it in temporary directory:

To verify and understand how your files are loaded into the package, use the restore command to list the contents of your bff image:

```
restore -Tvqf bffimage
```

For example:

```
restore -Tvqf devices.fcp.disk.IHVModel.1.1.0.0.bff
```

To extract your package into a temporary directory so you can verify your files are there:

```
mkdir /tmp/yourpackge
cd /tmp/yourpackage
copy bffimage /tmp/yourpackage
restore -xvqf bffimage
```

For example:

```
restore -xvqf devices.fcp.disk.IHVModel.1.1.0.0.bff
```

Verify your usr/lib/methods/IHV.cat file is in the packge.

To find your *.odmadd files, etc, you will need to extract the files out of liblpp.a found under ./usr/lpp/packagename/liblpp.a by using the command ar -xv liblpp.a

For example:

```
cd /tmp/yourpackage/usr/lpp/devices.fcp.disk.IHVModel
ar -xv liblpp.a
```

2. Test your installation package

Test your package install using both SMIT (smitty) and the installp command. For smit, use the "install" fastpath, as in "smitty install" and navigate through the menus to install and remove the package that you have created.

For testing on the command line, the "installp" command is used. The installp command has many flags and several different ways that it may be invoked. One possibility is

```
installp -agqwX -d <directory> <filesetname>
```

The flags shown have the following meaning:

-a Applies one or more products or updates
-g Automatically install or commit, respectively, any software products or updates that
 are requisites of the specified software product.
-q specifies quiet mode, which suppresses the prompt for the device
-w Does not wildcard FilesetName
-X Attempts to expand any file systems where there is insufficient space to do the installation

There is also a -c "commit" flag on installp. But since this document only deals with install images, the commit flag is not relevant as it applies only to update images.

3. Test package Uninstall

Uninstall your package with SMIT and with installp on the command line and verify your ODM and language file have been deleted.

When uninstalling your package, you can only use your fileset name on the command line, which in your case, will completely uninstall the package since there is only one fileset in your package.

One possible invocation of installp that will uninstall your package is:

```
installp -ug filesetname
```

where

-u  Removes the specified software product and any of its installed updates from the system

For example:

```
installp -ug devices.fcp.disk.IHVModel.rte
```

Reinstall your package and verify your ODM and language file have been readded.

# 11 Useful Commands

Below are examples of some commands you may find useful for displaying ODM entries and displaying device information on your system.  Refer to their man pages or the online AIX publications for further details about the AIX commands used in these examples.

4.  To retrieve ODM entries.

Use the odmget command to retrieve ODM entries.  A query may be specified with odmget to narrow down the retrieved stanzas.  For example, a device's uniquetype value may be used to query the database for objects specific to a device.

To retrieve all PdDv, PdAt, and PdAtXtd entries for a specific device type:

```
odmget -quniquetype=disk/fcp/ihvdisk PdDv

odmget -quniquetype=disk/fcp/ihvdisk PdAt

odmget -quniquetype=disk/fcp/ihvdisk PdAtXtd
```

5.  To determine a device's uniquetype.

Use the odmget command to retrieve the stanza from CuDv by name:

```
odmget -qname=hdisk3 CuDv
```

The PdDvLn field contains the uniquetype

6. Add ODM entries that are created in stanza format in a text file to the ODM database. odmadd is used to add ODM entries.

**odmadd <filename>**

Where <filename> is the name of an ASCII text file with ODM stanzas. The file name must end with ".odmadd"

7. Delete ODM entries.

odmdel is used to delete ODM entries. The odmdel command takes a query like odmget. The ODM database to delete the entries from is specified using the -o flag.

For example, to delete all PdDv, PdAt, and PdAtEtd entries for a specific uniquetype, do the following.

**odmdelete -quniquetype=disk/fcp/ihvdisk -o PdDv**

**odmdelete -quniquetype=disk/fcp/ihvdisk -o PdAt**

**odmdelete -quniquetype=disk/fcp/ihvdisk -o PdAtXtd**

To delete one attribute, e.g max_transfer:

**odmdelete -q "uniquetype=disk/fcp/ihvdisk AND attribute=max_transfer" -o PdAt**

8. To list all disks on the system

**lsdev -Cc disk**

9. List attributes of one disk, e.g hdisk2:

**lsattr -El hdisk2**

7. Discover and configure new devices.

**cfgmgr**

The scope of the configuration may be limited with the -l flag. To discover and configured devices attached to fcs0, for example.

**cfgmgr -l fcs0**

The cfgmgr command will configure all descendants of the specified device. For example, the command above will configure the fcs0 device, the children of fcs0 (such as fscsi0) and the children of fscsi0, which may include numerous hdisks.

10. To configure and unconfigure a single device.

The mkdev command can be used to configure a single device. Examples below:

**mdkev -l fcs1**
**mkdev -l fscsi1**
**mkdev -l hdisk3**

The rmdev command can be used to unconfigure single devices, or to recursively remove devices:

**rmdev -l hdisk3**
**rmdev -l fscsi1**
**rmdev -l fcs1**

The -R flag tells rmdev to unconfigure all descendant devices.

**rmdev -l fcs1 -R**

11. To determine if a package has been installed on your system use the lslpp command.

To display a particular package:

lslpp -l devices.fcp.disk.rte

# 12 FAQ (Frequently Asked Questions)

1) Can I add new vendor specific attributes (PdAt's) for my new device type I am adding to the system?

Answer: No. The configuration methods used to query the database during device discovery are not vendor supplied, so the AIX configuration methods provided on the system would not know about the new attributes.

2) If I already have added a Fibre Channel device and I want to add an iSCSI device, what is the best way to approach this?

Answer:

1. Retrieve all of your PdDv, PdAt, and PdAtXtd entries for your Fibre Channel device by either using odmget or starting with your *.odmadd file you created for your Fibre Channel device.

   Change every uniquetype field for each entry, substituting iscsi for fcp.

2. Determine the remaining missing iSCSI attributes (both required and optional) by reviewing the lists already given in this document, and create them by either doing an odmget on the "other" iSCSI device type disk/iscsi/osdisk.

   Go through your created attributes, removing the ones that are required for Fibre Channel only, and modifying your default values of the remaining ones to reflect your iSCSI device.

3. Update your PdDv to reflect your iSCSI device.

   Verify your message catalog file, IHV.cat already contains all of the messages for your new attributes. If it does not, you will have to modify your IHV.cat, update your original Fibre Channel package with the new file, and make your new iSCSI package have a requisite on your Fibre Channel package. See note below:

   Note: Any given file (full path included) can only be included in one fileset in AIX. So if you already have created a package containing the file /usr/lib/methods/IHV.cat, you will either need to use a different name for this package's catalog file, or update your original package's IHV.cat file and create a requisite to the original fileset in this package

3) What is needed to queue multiple commands to a device? I set queue_depth to a value greater than 1 for a disk, I observe (via the device or analyzer traces) that AIX is never queuing more than 1 SCSI CDB at a time to a LUN. What is required to get queuing to work for a (disk) LUN on AIX?

Answer: For FC and iSCSI, AIX requires that the LUN support NACA=1 before it will enable command queuing. AIX considers a device to support NACA=1 if the following 3 assertions are true for the device:

1. The ANSI version must be 3 or larger. ANSI version is found in byte 2 of Standard Inquiry data.
2. The NormACA bit must be 1. The NormACA bit is found in bit 5 of byte 3 of Standard Inquiry data.
3. The DQue bit must be 0. The DQue bit is in the Control Mode Page, page 0x0A, bit 0 of byte 3. If this bit is changeable, the mode data ODM must be included to ensure DQue is set to 0.

If all three of those conditions are true, AIX will queue commands to the device. If any one of the conditions is false, then AIX will never issue more than one SCSI CDB at time to that lun.

4) Does AIX support queuing to tape devices?

Answer: No. AIX only supports queuing to disk devices.

6) After installing my ODM on a boot disk and rebooting, AIX does not boot. What happened?

Answer: Without vendor-specific ODM installed, a disk may be recognized as an “MPIO Other” disk. By default, “MPIO Other” disks have the reserve policy set to use SCSI-2 reserves (i.e. the “reserve_policy” attribute shown by “lsattr –E –l hdiskX” will have a value of “single_path”). If vendor specific ODM is installed and it indicates that no reservations are to be used (e.g “reserve_policy” is set to “no_reserve”), then on reboot the disk may experience a reservation conflict. This will prevent AIX from booting if the reservation conflict occurs on the boot disk. To prevent this, the reservation must be cleared prior to rebooting AIX.

# Appendix A - Unique_id Additional Details

This appendix describes the syntax of the unique_id PdAt attribute and how it is used to create device's UDID value. Examples of a unique_id PdAt entry can be found in the "UDID Generation Details" subsection.

## The Unique Identifier

The "unique_id" attribute has three purposes. The primary purpose is to indicate that the device is capable of supporting multiple paths to it (e.g. it is an MPIO capable device). If the "unique_id" attribute is not present in a device's predefined configuration data, the device cannot support multiple paths to it. The second purpose is to provide a location in which to store an MPIO capable device's UDID value (when the attribute is created in the CuAt object class).

The final purpose is to provide information to aid in the generation of the device's UDID value. In the first release of MPIO, the only supported MPIO devices are SCSI and fiber channel devices. Both of these sets of devices have "inquiry data" that can be obtained from the device. This "inquiry data" has some standard information and some optional information. The generation of the UDID uses information from both areas. The PdAt object of the "unique_id" attribute contains information how the UDID is constructed from the "inquiry data". For a complete description, refer to the Design Specification for MPIO Device Driver for complete details about the mechanism used to construct a UDID for SCSI and fiber channel devices.

In the future, if other device types are added to the MPIO support list, the information in the 'PdAt' may be different for those future device types. The definition here applies to iSCSI and Fiber Channel types of devices only.

The "unique_id" attribute has the following characteristics/requirements:

The 'uniquetype' field identifies the target (child) device to which the attribute belongs.
The 'type' field must have value of "R".
The 'attribute' field must have the value of "unique_id".
The 'deflt' field should be an empty string.
The 'values' field identifies locations in the inquiry data to acquire certain pieces of information required to construct the UDID of the device.
The 'generic' field should identify this attribute as displayable("D")
The 'nls_index' field should identify the message number that correctly describes the attribute.

NOTE: If the "unique_id" attribute is not present in the target device's ODM data, the device will not be configured or treated as an MPIO capable device.

## UDID Generation Details

This section describes how a UDID is generated for a SCSI, Fiber Channel or iSCSI disk device. This section also describes how the unique_id ODM attribute will be used to generate a UDID. The UDID format is:

UDID=UDID_LEN + sec1_len + sec1_data + sec2_len + sec2_data + sec3_len + sec3_data+UDID_TYPE

The following is a description of how the UDID generation algorithm, implemented within the device specific methods, builds each of the pieces of the UDID. There may be alternate algorithms supplied in future releases to handle devices whose UDID cannot be generated by the following algorithm.

UDID_LEN - The UDID_LEN will consist of two uppercase hexdecimal characters representing the length in bytes of the UDID, not including the two bytes of the UDID_LEN. The UDI_LEN has a maximum length of 253.

sec1_len - The sec1_len will consist of two uppercase hexdecimal characters and will be set to the length in bytes of the sec1_data.

sec1_data -The sec1_data is the device serial number and is defined by the unique_id ODM attribute. The unique_id ODM attribute defines one or more device serial number substrings which when concatenated together form the device serial number.

If the unique_id ODM attribute value is "" or if all substrings read from the code pages are "", the UDID generation routine will fail and return an error to the calling routine.

Each time a code page is read attempting to acquire a substring the ASCII representation of the hexidecimal code page number will be catenated to the sec1_data followed by the substring. The sec1_data must consist of ASCII characters and may not contain a 0x00 value.

The following describes in more detail the relationship between the unique_id ODM attribute and the device serial number ( sec1_data).

The unique_id value PdAt field will consist of one or more "sets" of values, comma delimited. The number of comma delimited sets is limited by the size of the unique_id ODM attribute which is 255. Each "set" is made up of two or three hex two digit values which represent the code page,offset into a page and the length of data to read. The unique_id "sets" will be used to create a unique device serial number. Each of these "sets" defines a substring of the device serial number ( sec1_data).

The following is an example to help explain the format -
unique_id= OOLLPP,OOLLPP,OOLLPP
or
unique_id= OOLL,OOLL

where:
OO - the offset in bytes within a code page
LL - the length in bytes of the value to be read
PP - the code page number

NOTE: if the PP value is not specified the standard inquiry data will be used

sec2_len

The sec2_len will consist of two uppercase hexdecimal characters and will be set to the length in bytes of the sec2_data.

sec2_data

The sec2_data consists of the device product id from the device standard inquiry page. The product id is located at an offset of 16 bytes within the inquiry page and has a length of 16 bytes in the standard inquiry page. If the product id has any trailing spaces they will be stripped. The sec2_data must consist of ASCII characters and may not contain a 0x00 value.

sec3_len

The sec3_len will consist of two uppercase hexdecimal characters and will be set to the length in bytes of the sec3_data.

sec3_data

The sec3_data consists of the device vendor id from the device standard inquiry page. The vendor id is located at an offset of 8 bytes within the inquiry page and has a length of 8 bytes.  If the vendor id has any trailing spaces they will be stripped. The sec3_len will be set to the length in bytes of the sec3_data. The sec3_data must consist of ASCII characters and may not contain a 0x00 value.

UDID_TYPE

The UDID_TYPE will be set to the device subclass descriptor in the PdDv object for the device. Only "scsi", "fcp", and "iscsi" will be supported in the first release of MPIO. A 0x00 value will be appended to the UDID_TYPE to terminate the UDID string. The following is an example of the algorithm. The ODM for the example device is included to help show the relationship between the unique_id ODM attribute and the device serial number ( sec1_data).

```
PdAt:
uniquetype = "disk/fcp/my_disk"
attribute = "unique_id"
deflt  = ""
values = "040880,040883"
width  = ""
type  = "R"
generic = "D"
rep  = "nl"
nls_index = 100
```

In this example, the product id is PRODUCTID, the vendor id is VENDORID, the device serial number from code page 0x80 was 0x00 and the volume identifier from code page 0x83 was DEVICESERIALNUMBER.

UDID=31168083DEVICESERIALNUMBER09PRODUCTID08VENDORIDscsi

Note :

31 is the UDID_LEN in hex
16 is the sec1_len in hex
80 is code page 0x80, since no device serial number existed here there is no other  data
83 is code page 0x83

DEVICESERIALNUMBER was the value of the serial number found in code page 0x83
09 is the sec2_len in hex
PRODUCTID is the name of the product
08 is the sec3_len in hex
VENDORID is the name of the vendor.
scsi is the UDID_TYPE.
Note : when converting decimal values 11-15 to acsii hexidecimal digits uppercase A-F letters must be used.

A device with a device serial number that is not unique for the vendor/product id will be detected and configured as a MPIO capable device but if other devices are detected with this same UDID, then these two physically different devices will be detected and configured as a single MPIO device. This may cause system or data integrity errors. So it is important that the fields specified by the unique_id attribute do uniquely identify the device.

# End of Document