Listing of some AIX v. 6 (and v. 5) keypoints for Exam 223 (formerly known as 222).

Version: 3.8
Date: 28 October 2010

By: Albert van der Sel - Antapex Technologies B.V.

Contents:

Chapter 1. Overview AIX volume and filesystem management.

1.1 Physical Volumes (disks) and Volume Group.

The above terms, all belong in the realm of a "Logical Volume Manager", or LVM.

In AIX, it's common to use a "Logical Volume Manager", or LVM, to cross the boundaries posed by traditional disk management.
Traditionally, a filesystem was on a single disk or on a single partition of that disk (slice).
With a LVM, we can create "logical volumes" (=filesystems) which can span several disks.

The LVM has been a feature of the AIX operating system since version 3, and it is installed automatically with the Operating System.

The "old" (standard non-LVM) way to create a filesystem on UNIX, say on a "older" Solaris version, say v. 8:

Although the following is not part of an AIX description at all, it is instructive to see how "old" filesystem management was done in the past. Say, we wanted to create a filesystem on a new disk, in Solaris v.8:
(and suppose we don't have a LVM like Veritas)

Example:

Suppose we have this (disk) device "/dev/rdsk/c0t3d0s2".
This is the object "c0 - controller 0, t3 target 3, d0 lun 0, s2 - slice 2", where slice 2 means "the whole diskpartition".
Now we proceed as follows:
Format that disk:

```
# format /dev/rdsk/c0t3d0s2
FORMAT MENU:
(we do our stuff here)
format>label
(we do our stuff here)
format>partition
(we do our stuff here)
```

When we are done with partioning, we can, for example, create an UFS filesystem like in the following command:

```
# newfs /dev/rdsk/c0t3d0s0
```

Now, this example may not strike you as relevant at all, since this document is about AIX.
But hopefully you can see that we only deal with one disk at the time. Here, you can create a filesystem only on a partition of a disk, and at most covering that whole disk.

- We do not have a way of creating a (large) filesystem that uses multiple disks.
- Also, it's not easy to implement a form of redundancy or protection, like mirroring the filesystem.

Because of the above two reasons, an LVM will almost always be used on modern Unix systems.

1.1.1. The concept of "Volume Group".

When you have installed AIX, you will always have (at least) one Volume Group, called "rootvg", which contains *all to the Operating System belonging filesystems*.

A Volume Group (hereafter called "VG"), consists of one or more disks.
In AIX, a "disk" is called a Physical Volume (herafter called "PV").

The LVM makes it possible to create the VG from one or more (for example, three) physical disks, as if we combine the disks into that Volume Group.

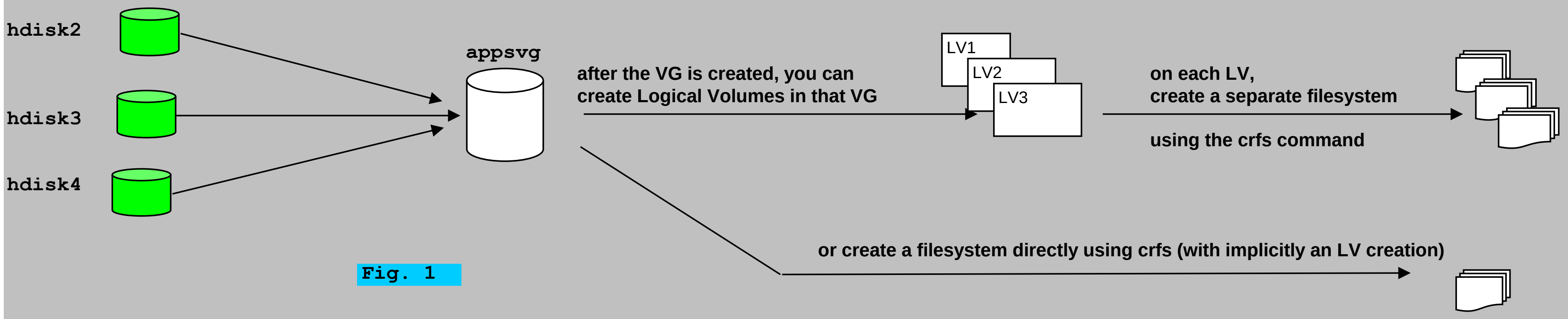

hdisk2
hdisk3
hdisk4
appsvg
after the VG is created, you can create Logical Volumes in that VG
LV1
LV2
LV3
on each LV, create a separate filesystem
using the crfs command
or create a filesystem directly using crfs (with implicitly an LV creation)

Fig. 1

At the moment, that you would enter the command to create the Volume Group (with a certain name like e.g. "appsvg"),
from the three physical disks, the following happens:
The physical disks get's "partitioned" into contiguous equal-sized units of space called "Physical Partitions" (PP),
Don't think of partitions in the way "*like disk partitions in the PC world*". No, the "partitions" here,
are sort of "extends", a number of diskblocks, which number depends to the PP size you have specified.
32M, 128M etc...
You can control the size of the Physical Partition, by specifying it as a command option, like e.g. 4M, 16M,

From now on, we just have a "sea" of Physical partitions to deal with. They are "contained" in the Volume Group.

But how is that possible? Suppose the system reboots. How does the LVM "knows" which PV's (thus which disks) belong
to which VG's?

There are two answers:

1. All Volume Group and Physical Volume information, get's stored in the ODM. Especially on boot,
   the system then knows how to identify the VG's and corresponding PV's.
2. All Physical Volumes belonging to a VG, have a socalled "Volume Group Descriptor Area" (VGDA).
   The VGDA, located at the beginning of each physical volume, contains information that describes all
   the LV's and all the PV's that belong to the VG of which that PV is a member.
   The VGDA makes a VG selfdescribing. An AIX System can read the VGDA on a disk, and from that, can
   determine which PV's are part of this VG.

1.1.2 The "lspv" command:

Before we will create a VG, let's take a look on how we can list the PV's (the disks)
that are known to our system.
Ofcourse, we already have seen the "lsdev" command, which can show use which disks
are known to the ODM.

But in the context of LVM, the "lspv" (list physical volumes) command is much better.
Not only shows that command all disks that are known to the system, but also which of them already belong
to a VG, and which PV's are still "free" to be used.

Here are a few examples:

```
P550LP1:/root #  lspv

hdisk0          00c723ba50b1f7f0                    rootvg          active
hdisk1          00c723ba6ee6c820                    rootvg          active
hdisk2          00c723ba6f1ec7c1                    oravg           active
hdisk3          00c723ba6f1f5319                    dumpvg          active
hdisk4          00c723ba6f1ec9b7                    oravg           active
hdisk5          00c723ba6f1f5507                    dumpvg          active
```

So in upper example, on the AIX system P550LP1, the PV's hdisk2 and hdisk4 are part of the VG called "oravg".

```
P550LP3:/root #  lspv

hdisk0          00c723ba50b1f7bc                    rootvg          active
hdisk1          00c723ba6ee6c730                    rootvg          active
hdisk2          00c723ba6f1ec8c3                    none
hdisk3          00c723ba6f1f5512                    none
```

So, on the AIX system P550LP3, the PV's hdisk0 and hdisk1 belong to the "rootvg" Volume Group.
But hdisk2 an hdisk3 are still not assigned to any VG, and might be used to create one.

Also, note the 16 digit unique identifier in the second column. This is called Physical Volume Identifier(PVID).
This ID uniquely identifies a disk on your system. Without it, a Physical Volume can not be used under LVM.

Note: actually the PVID is somewhat longer than you see with the lspv command.
We can view the complete PVID, and many other details, using the lsattr command, like in this example:

```
# lsattr -El hdisk2
PCM             PCM/friend/sddpcm                                      PCM                                     True
PR_key_value    none                                                   Reserve Key                             True
algorithm       load_balance                                           Algorithm                               True
dist_err_pcnt   0                                                      Distributed Error Percentage            True
dist_tw_width   50                                                     Distributed Error Sample Time           True
hcheck_interval 20                                                     Health Check Interval                   True
hcheck_mode     nonactive                                              Health Check Mode                       True
location                                                               Location Label                          True
lun_id          0x0                                                    Logical Unit Number ID                  False
lun_reset_spt   yes                                                    Support SCSI LUN reset                  True
max_transfer    0x40000                                                Maximum TRANSFER Size                   True
node_name       0x50050768010029c8                                     FC Node Name                            False
pvid            00cb5b9e66cc16470000000000000000                       Physical volume identifier              False
q_type          simple                                                 Queuing TYPE                            True
qfull_dly       20                                                     delay in seconds for SCSI TASK SET FULL True
queue_depth     20                                                     Queue DEPTH                             True
reserve_policy  no_reserve                                             Reserve Policy                          True
rw_timeout      60                                                     READ/WRITE time out value               True
scbsy_dly       20                                                     delay in seconds for SCSI BUSY          True
scsi_id         0x611013                                               SCSI ID                                 False
start_timeout   180                                                    START unit time out value               True
unique_id       33213600507680190014E30000000000001E204214503IBMfcp Device Unique Identification           False
ww_name         0x50050768014029c8                                     FC World Wide Name                      False
```

So, the complete PVID is "padded" with 16 zeroes on the "right" side.

Note: if the lspv command does not show a PVID for a certain PV, let the system create one.
You can use the "chdev" command for that, like in this example:

```
# chdev -l hdisk2 -a pv=yes
```

Note: the "P550LP1:/root #" string, is just the used "prompt" here. A prompt can be as simple as just "#" (for root),
or "$" or "%" (for a regular user), or it can be a more elaborate expression like for example containing the Hostname
(here P550LP1), and current path (here it is /root).

Other examples using "lspv" with some important flags:

The syntax is:

```
lspv              (just use lspv, as we have seen above)
lspv [ -L ] [ -P ] [ -l | -p | -M ] [ -n descriptorphysicalvolume] [ -v volumegroupid] physicalvolume
```

```
# lspv hdisk23                # shows summaries as well as id's

PHYSICAL VOLUME:    hdisk23                  VOLUME GROUP:     oravg
PV IDENTIFIER:      00ccf45d564cfec0 VG IDENTIFIER     00ccf45d00004c0000000104564d2386
PV STATE:           active
STALE PARTITIONS:   0                        ALLOCATABLE:      yes
PP SIZE:            256 megabyte(s)          LOGICAL VOLUMES:  3
TOTAL PPs:          947 (242432 megabytes)   VG DESCRIPTORS:   1
FREE PPs:           247 (63232 megabytes)    HOT SPARE:        no
USED PPs:           700 (179200 megabytes)
FREE DISTRIBUTION:  00..00..00..57..190
USED DISTRIBUTION:  190..189..189..132..00
```

```
# lspv -p hdisk23             # shows free/used over the PP ranges and the distribution of the disk

hdisk23:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-22    used    outer edge    u01                 jfs2       /u01
 23-190   used    outer edge    u02                 jfs2       /u02
191-379   used    outer middle  u01                 jfs2       /u01
380-568   used    center        u01                 jfs2       /u01
569-600   used    inner middle  u02                 jfs2       /u02
601-700   used    inner middle  u03                 jfs2       /u03
701-757   free    inner middle
758-947   free    inner edge
```

```
# lspv -l hdisk0              # shows all LV's and LP/PP's and mountpoints on that disk

hd5                    1     1     01..00..00..00..00    N/A
prjlv                  256   256   108..44..38..50..16   /prj
hd6                    59    59    00..59..00..00..00    N/A
fwdump                 5     5     00..05..00..00..00    /var/adm/ras/platform
hd8                    1     1     00..00..01..00..00    N/A
hd4                    26    26    00..00..02..24..00    /
hd2                    45    45    00..00..37..08..00    /usr
hd9var                 10    10    00..00..02..08..00    /var
hd3                    22    22    00..00..04..10..08    /tmp
hd1                    8     8     00..00..08..00..00    /home
hd10opt                24    24    00..00..16..08..00    /opt
```

1.1.3 The "mkvg" command: Creating a Volume Group:

Let's create a VG called "appsvg" on the system "P550LP3" of the former section.
We know that on this system, hdisk2 and hdisk3 are not assigned to a VG, so, they can be used to create "appsvg".

To create a VG, you use the "mkvg" (make volume group) command.
In it's simplest form, the command is:

```
mkvg -y <name_of_volume_group> -s <partition_size> <list_of_hard_disks>
```

So, let's us try the following on P550LP3, using both hdisk2 and hdisk3:

```
P550LP3:/root #  mkvg -y appsvg -s 64 hdisk2 hdisk3
```

This command creates the "appsvg" VG, using hdisk2 and hdisk3. We do not *have* to use both hdisk2 and hdisk3.
Typically, you would use multiple disks, if you want to create a large filesystem that spans multiple disks,
or if you want to implement redundancy by using mirroring.

Note:
Yes, creating a VG is only part of the process. Once a VG is created, you should create one or more "Logical Volumes",
on which you can create filesystems. So, after creating VG's, you are only halfway there.

In the above example, I used a Physical Partition size of 64 MB.
You can use a "power of 2" as a Physical Partition (PP) size, that is for example, 1,2,4,8,..64,128,256.. MB
If you leave out the "-s <size>, the system will determine a proper PP size (and -t factor: explained later).

Is the PP size choosen important? Yes, there are two considerations:

- It might be a factor in performance considerations.
If you just have a few local disks, I would say it does not matter too much. For example, you could choose 64MB or 128MB.
But suppose you have a local expensive storage system, say with 128 disks, there are recommendations on
how you best choose a PP size for a certain purpose (e.g. as Storage for a Database, using some RAID implementation).
So, let's say that when you use the "fancy" storage systems, advice from Storage experts or from the manufacturer
is recommended. It's just part of your planning and implementation process.
- It can impose contraints on how many disks a VG can contain.
Later more on this.

Here are some other examples:

```
P550LP3:/root #  mkvg -y appsvg -s 64 hdisk2                               # here we use only 1 PV
P550LP3:/root #  mkvg -y datavg -s 64 hdisk3                               # here we use only 1 PV

P570LP14:/root #  mkvg -y datavg -s 128 hdisk3 hdisk4 hdisk5               # here we use 3 PV's
```

For easy reference, here is the full mkvg syntax: (you don't need to memorize all options)

```
mkvg [ -B ] [ -t factor ] [ -S [ -v logicalvolumes ] [ -P partitions ] ] [ -C ] [ -G ] [ -f ] [ -i ] [ -I ]
[ -c ] [ -x ] [ -L ltgsize ] [ -n ]
[ -s size ] [ -V majornumber ] [ -y volumegroup ] [ -M y|s ] [ -p mirrorpool ] physicalvolume_list
```

1.1.4 The "mkvg4vp" command: Creating a "vpath" Volume Group:

In some Storage configurations, "SDD" is used. SDD means "Storage Multipath Subsystem Device Driver".
Mainly, SDD provides multipath (indeed thus "multiple paths") support for a host AIX system
to attach to a storage device (for example, through multiple fibercards).

The Physical Volumes we have seen so far, were called "hdiskn", like hdisk0, hdisk1 etc..
But in the "SDD world", the disks are named "vpath1", "vpath2" etc..

In this case, an optimised "mkvg" command should be used. This command is "mkvg4vp", as short for "mkvg for vpath".

Here are a few examples:

```
P570LP14:/root #  mkvg4vp -B -t 32 -s 4 -y DB01_RECOV_VG1 vpath4 vpath10
```

Here, the VG is called "DB01_RECOV_VG1", and as you can see, two extra flags were used, namely the "-B" and "-t" flags.
Don't worry about those extra flags right now.

The mkvg4vp is optimized for using advanced flags in creating a VG. For a simple VG, the command "mkvg" can be
used as well. But for advanced options like an 'enhanced concurrent VG', we should use the mkvg4vp command.

1.1.5 Listing VG's:

Listing Volume Groups:

Let's see how we can obtain a listing of the Volume Groups that exists on your system.
It probably won't surprise you; the command is "lsvg" (list volume group)

So, here is an example:

```
P550LP3:/home/alberts #  lsvg
rootvg
appsvg
```

Indeed, we see our VG's listed, including the new "appsvg" VG, that we created in the former section.

Listing the File systems (or Logical Volumes) on a Volume Group:

Although we still need to introduce the concept of "Logical Volume", and the filesystem you can create
on a Logical Volume, at this time I already want to show the use of the "-l" flag, by which
you can see which LV's (and which corresponding filesystems) reside on the VG.

```
lsvg -l <Volume_Group_Name>
```

```
# lsvg -l datavg

datavg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv01             jfs2log     1     1     1    open/syncd    N/A
lv02                jfs2        240   240   1    open/syncd    /data
lv03                jfs2        384   384   1    open/syncd    /dump
lv07                jfs2        256   256   1    open/syncd    /was6
```

So, in this example, the VG "datavg" contains 3 filesystems: /data, /dump, and /was6.
Also, observe that a "journaled log" exists in any VG. That is because the native AIX filesystems are JFS or JFS2,
which are "journaled" filesystemtypes (meaning that a write-ahead log exists, to log metadata changes).

Listing Technical specs (like the VGID) from a Volume Group:

Use the "lsvg <Volume_Group_Name>" to retrieve technical specs from a specified VG, like so:
Notice that you can find the VGID (Volume Group ID) with this command.

```
# lsvg oravg

VOLUME GROUP:   oravg                    VG IDENTIFIER:  00ccf45d00004c0000000104564d2386
VG STATE:       active                   PP SIZE:        256 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      1894 (484864 megabytes)
MAX LVs:        256                      FREE PPs:       492 (125952 megabytes)
LVs:            4                        USED PPs:       1402 (358912 megabytes)
OPEN LVs:       4                        QUORUM:         2
TOTAL PVs:      2                        VG DESCRIPTORS: 3
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     2                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
LTG size:       128 kilobyte(s)          AUTO SYNC:      no
HOT SPARE:      no                       BB POLICY:      relocatable
```

Listing a Voluming Group, showing disks, Total free/used PP's, and distribution:

```
# lsvg -p informixvg

informixvg
PV_NAME       PV STATE     TOTAL PPs     FREE PPs     FREE DISTRIBUTION
hdisk3        active       542           462          109..28..108..108..109
hdisk4        active       542           447          109..13..108..108..109
```

1.1.6 Adding and removing disks to and from a Volume Group:

You can add a PV (a disk) to an existing VG. In this case, the PV inherits the properties
of the VG, like for example the PP size. Also, a new disk obviously get's an VGDA.

You can also remove a PV from an existing VG. If you have removed (or reduced) the last PV,
the VG cease to exist.

Adding a PV (disk) to a Volume Group: extendvg

It is easy to add a disk, or a number of disks, to a VG.
Simply use the "extendvg" command:

```
extendvg <Volume_Group_Name> <List_of_PV>
```

Examples:

```
# extendvg newvg hdisk23
# extendvg appsvg hdisk5 hdisk8
```

Removing a PV (disk) from a Volume Group: reducevg

It's also easy to remove a disk from a VG, although here you must be carefull to select the right PV.

WARNING: if the disk contains LV's (and thus filesystems), or is part of a LV spanning multiple disks, you better KNOW EXACTLY WHAT YOU ARE ABOUT TO DO.

To remove a PV from a VG, use:

```
reducevg <Volume_Group_name> <PV_name>
```

Examples:

```
# reducevg myvg hdisk23
```

If the VG contains filesystems, use the -d flag, like in:

```
# reducevg -d myvg hdisk23
```

Important: if you only want to remove a disk from a VG, and this disk contains data, and the other disks in the VG have still space to hold that data, then use the migratepv command, before deleting the disk from the VG.

Suppose VG datavg consists of hdisk2, hdisk3 and hdisk4.
Now suppose that you want to remove hdisk4 from datavg, while preserving the data by migrating that data to the other disks. Then use the "migratepv" command like in:

```
# migratepv hdisk4 hdisk2 hdisk3
```

When you delete the last disk from the VG, the VG is also removed.

Varyon and Varyoff a VG:

You can "shutdown" a VG, so that it is inactive and the filesystems on that VG cannot be accessed by users.
For this, you can use the "varyoffvg" command.

The other way around, you can "open" a VG that was "closed" by using the "varyonvg" command.
All its resident filesystems are mounted by default if they have the flag "mount=true" in the "/etc/filesystems" file (more on that later).

Examples:

```
# varyoffvg appsvg
# varyonvg appsvg
```

Only "rootvg" (the VG containing the OS), is a bit hard to vary off.

1.2 Managing Logical Volumes and Filesystems on a VG:

1.2.1. Creating and Managing a Logical Volume:

- Creating a Logical Volume:

We know how to create VG's an how to manage them. Next, we want to create filesystems.
In AIX (and most other unixes with LVM's), you often create a "Logical Volume" first, before
you create a filesystem on that Logical Volume (LV).

Note: as you will see later, the "crfs" command (used to create a filesystem), does not perse
need a LV to exist beforehand. You can just specify a Volume Group (along with other options), and AIX
will create a filesystem in that Volume Group (and creates as system generated Logical Volume in the process).
Because it's important to know how to handle LV's, we will review how to create a filesystem, using an
existing LV, first. That's why we need to know how to create and handle a LV.

A Volume Group itself, resembles nothing else than a "container" with Physical Partitions (PP's).
With a LV, you create a "structure" inside that container, with certain properties like
how many copies (most often 2) of a PP you want, for safety (redundancy).
So, the LV uses an abstract Partition, called "Logical Partition" (LP), which is actually equivalent
to a Physical Partition and its optionally redundant copies (of that Physical Partition).

Since a VG is created (usually) from multiple disks, a Logical Partition, could be equivalent
to a PP on one disk, and a PP on another disk. So, in this example, there are 2 PP's, which are mirrored copies
and which are represented by one Logical Partition.
This way, you have implemented mirroring, on a PP level.

It is not required to have multiple copies of PP's. If a Volume Group consists of only one disk,
it is useless to implement PP mirorring, because you cannot implement safety (real redundancy) here.

So,
A LP maps to (at least) one PP, and is actually the smallest unit of allocatable space in creating a LV.
But, for safety reasons, usually a LP is mapped to 2 PP's (which are copies on different disks).

In a former section, we created the "appsvg" Volume Group. The PP size choosen was 64M.
Below is the command that we used:

```
P550LP3:/root #  mkvg -y appsvg -s 64 hdisk2 hdisk3
```

In this case, we used 2 disks, so here we are able to create a Logical Volume, where each LP
has 2 equivalent PP's, where one PP reside on hdisk2 and a copy of that PP reside on hdisk3.

In it's most simple form, the command to create a LV looks like

```
mklv -y <Logical_Volume_Name> -c <No_of_Copies_of_PPs> <Volume_Group_Name> <No_of_LPs>
```

So, here is a typical command to create a Logical Volume. Here the LV is named "lv05":

```
# mklv -y lv05 -c 2 appsvg 100
```

- The command to create a LV is "mklv" (make logical volume).
- With "-y" you can specify the name of your LV.
- With "-c" you can specify how many PP copies you want for each LP. In this case, we have choosen for 2 copies, which is about right since we have 2 disks in the Volume Group.
  So, here we have a "mirrored LV", which actually means we have 2 mirrored PP's per LP.
- In the command, you need to specify in which VG you want to create the new LV. Here it is "appsvg".
- Also, you need to size your LV, and in the above example we want 100 mirrored PP's.

Now, what about the size of our new LV in the above example.
The lv05 Logical Volume, has a size of 100 x 64MB = 6400 MB.
But it's mirrored, so, there are 100 PP's on hdisk2 and 100 PP's on hdisk3.
This is the way it goes with mirrored stuff. Although 200 PP's are used, thus 12800 MB is in use,
each PP is mirrored, so in effect you only have 100 PP's to store data on.

Here are a few other examples:

```
# mklv -c 3 vg03 9                    # since we did not provide a LV name, the system will generate one.
# mklv -y lv02 vg03 10                # here the default of 1 copy is implemented. So there is only one PP per LP.
```

Important: whether you should implement mirroring (or similar) by using the '-c' flag, just depends
on the Storage where you are going to create the LV on. Suppose you have a LUN from a SAN, and that
storage is on RAID 5 (or so) anyway, then you do not need to mirror. So, it just depends on the Storage that
is available to your system. In larger systems, it's very common that the sysadmin and the storage admin,
will both contribute to find the best options.
On smaller systems, using local disks, or using a local diskcabinet, you probably do want to use mirroring.

For easy reference, here is the full mklv syntax: (you don't need to memorize all options)

```
mklv [ -a position ] [ -b badblocks ] [ -c copies ] [ -C stripewidth ] [ -d schedule ] [ -e range ] [ -i ] [ -L label ]
 [ -m mapfile ] [ -o y / n ] [ -r relocate ] [ -s strict ] [ -t type ] [ -T O ] [ -u upperbound ] [ -v verify ]
[ -w mirrorwriteconsistency ] [ -x maximum ] [ -y newlogicalvolume | -Y prefix ] [ -S stripsize ] [ -U userid ]
[ -G groupid ]  [ -P modes ] [ -p copyn=mirrorpool ] volumegroup number [ physicalvolume ... ]
```

- Removing a Logical Volume:

To delete a LV, use the "rmlv" command, like in the following example:

```
# mklv -y lv05 -c 2 appsvg 100
```

Now, let's delete lv05:

```
# rmlv lv05
Warning, all data on logical volume newlv will be destroyed.
rmlv: Do you wish to continue? y(es) n(o) y
#
```

- Extending a Logical Volume:

If a VG still has free PP's, you can extend a LV with a number of LP's.
For example:

```
# extendlv lv05 20
```

If the LV used mirorred PP's, for example, the common number of 2 mirrored PP's per LP,
then 20+20 PP's are allocated.

1.2.2. Creating a File System:

- A few remarks on JFS and JFS2:

There are two "native" r/w filesystem types available in AIX: jfs and jfs2.
JFS is short for: "Journaled File System".
JFS was already available since AIX v3.1, and JFS2 was introduced with AIX v 5.1.
It's reasonably fair to say that JFS is seen mainly on systems using AIX up to v. 5.2.
Gradually, JFS2 took over, and it's "reasonably fair" to say that most systems since AIX 5.2 uses JFS2.
JFS2 is also called "Enhanced Journaled File System".

But AIX is not limited to those two filesytems, because if you want/must/need, you might use
another LVM and use a fileystem type like for example the widely used VxFS.

Both types of filesystems are "journaled filesystems", meaning that a writeahead logging "log" exists,
before metadata changes are actually applied to the data on the filesystems, enhancing recoverability after a crash.
Here, metadata is all info about a file, except the data itself.

JFS2 is optimized for use at a 64bit kernel.
JFS2 is the default filesystem in AIX 6.1
JFS and JFS2 filesystem types, can both be used at the same time in all versions as of v5.1.

With JFS2, it's not likely that you will run out of "inodes" of a filesystem, because they are added as needed.
Contrary, if you created a JFS filesystem of a certain size, the number of inodes is fixed.

There are many differences between JFS and JFS2, like maximum filesystem size, maximum file size etc..

	JFS	JFS2
Max Filesystem size	1 TB	32 TB (theoretically much larger)
Max File size:	63.876 GB (about 64GB)	16 TB (theoretically much larger)
inodes:	fixed, at fs creation	dynamic
Directory structure:	B tree	linear
Compression:	Supported	Not supported

In general, you might say that JFS2 should be preferred over JFS (there still are quite a few exceptions though).

Important is the fact that filesystem compression is supported on JFS, but not on JFS2 !
Really ! This fact has been spotted as an exam question.

(Here we do not mean the compression by tools as "compress", "gzip" etc.., but compression as an internal
function of the filesystem, that is, how "fragments" are stored in diskblocks).

- Creating filesystems:

Usually filesystems get's created by using:

```
1. Smitty          It's a reasonable friendly menu system (ascii, non graphical) by which you can
                   do almost all administration like adding useraccounts, configure networking, creating fileystems etc..

2. The "crfs" command (create filesystem)
3. The "mkfs" command (make filesystem)
```

For the commands, the syntaxes are:

crfs:

```
crfs -v VfsType { -g VolumeGroup | -d Device } [ -l LogPartitions ] -m MountPoint [ -n NodeName ]
[ -u MountGroup ] [ -A { yes | no } ] [ -p {ro | rw } ] [ -a Attribute=Value ... ] [ -t { yes | no } ]
```

Or in the most simple form:

```
crfs -v <Type_Of_Filesystem (jfs/jfs2)>  [-g <Volumegroup> -a size=] or [-d <Logical_Volume>] -m <The_Mountpoint_YouWant>
```

Do you notice that you can specify either a Volume Group to create the filesystem in,
or a Logical Volume to create the filesystem on (which exists in a VG).
So, you do not perse create a LV beforehand, but generally speaking it's better.
If you use the "-g <Volumegroup>", then AIX will create a LV for you, because a Filesystem is always related to a LV.

Examples:

```
# crfs -v jfs -d lv10 -m /data -a bf=true
```

In this example, the LV "lv10" already existed. We do not need to specify a size, because that was already
determined when we created the LV.
The jfs filesystem get's created, and afterwards it can be (per default) mounted as "/data".
Ofcourse, you can mount a filesystem on any mountpoint of your choice, but the "-m" flag will create
entries in the "/etc/filesystems" file, which is the default repository of filesystem information on AIX.

```
# crfs -v jfs2 -g rootvg -a size=381M -m /tftpboot -A yes -t rw
```

Note the difference in "-g VolumeGroup" or "-d LogicalVolume". In case of a LV, you do not need to specify the size,
because the LV is already of a certain size.

***Note: the "/etc/filesystems" file, has the same function as "/etc/fstab"
that can be found on many other unixes.***

mkfs:

```
mkfs [-b Boot] [-l Label] [-i i-Nodes] [-o Options] [-p Prototype] [-s Size] [-v VolumeLabel] [-V VfsName] Device
```

Or in it's most simple form:

```
mkfs -V <Type_of_Filesystem> -o options <Device>
```

Here are some important differences with the "crfs" command. With mkfs, you create a filesystem
on a device like "/dev/hd3" (a disk) or "/dev/lv05" (a Logical Volume expressed as a device name)
Also, note that there is no flag to specify a mountpoint beforehand.

Example:

```
# mkfs -V jfs2 /dev/lv01              # this decive is a LV
```

Why two commands?

On most other unixes, "mkfs" is the rough variant. Most of the time, a more easier interface is available
like the "newfs" command (which just uses the mkfs command below the surface).

This is not exactly the same in AIX. Both commands will generally do the same, but mkfs is more
equipped in doing special operations. Usually, sysdamins will use the "crfs" command.

Both commands can be used to create filesystems in any normal operation.
But in AIX, using "mkfs" a device like a LV, needs to exist.
And with "crfs", if you just specify the Volume Group, a device (a LV) will be created in the process.

1.2.3. The "/etc/filesystems" file:

Almost all unixes have a sort of repository (just an ascii file) which states which filesystems are present, and
which should be mounted at boottime, as well as some other "options" like if "fsck" should
check the filesystem etc..

In AIX, this is the "filesystems" file, which can be found in "/etc".
Here is an example of a part of such a file:

```
/data:
        dev             = /dev/lv03
        vfs             = jfs
        log             = /dev/loglv01
        mount           = true
        options         = rw
        account         = false

/apps:
        dev             = /dev/lv04
        vfs             = jfs
        log             = /dev/loglv00
        mount           = true
        options         = rw
        account         = false
etc..
```

This indeed a bit different format, compared to the /etc/fstab file found in many other unixes.
The advantage is, is that this one is easily editable (with vi).

1.2.4. Change the size of a filesystem using chfs:

The "chfs" command, changes attributes of a filesystem.

Especially it is usefull in increasing or decreasing the size of a filesystem.
Ofcourse, if a filesystem is increased, the associated LV will be adjusted as well.

The "-a" flag specifies the attribute you want to adjust.

Examples:

```
# chfs -a size=+128M /opt
# chfs -a size=+20G /data/udb
```

The + is interpreted as a request to increase the file system size by the specified amount.
As of AIX v5.3, you can shrink a JFS2 filesystems as well, using '-'.

1.2.5. The "lsfs" command:

Displays the characteristics of file systems.

(If you want to view the contents, just use the normal "ls" or "ls -al" command).

Syntax
```
lsfs [ -q ] [ -c | -l ] [ -a | -v VfsType | -u MountGroup| [FileSystem...] ]
```

To show all file systems in the /etc/filesystems file, enter:
```
# lsfs
```

To show all jfs filesystems, enter:
```
# lsfs -v jfs
```

-q flag:

The lsfs command displays additional Journaled File System (JFS) or Enhanced Journaled File System (JFS2)
characteristics if the -q flag is specified.

```
# lsfs -q /opt
Name Nodename Mount Pt VFS Size Options Auto Accounting
/dev/hd10opt -- /opt jfs 4128768 -- yes no
(lv size: 4128768, fs size: 4128768, frag size: 512, nbpi: 4096, compress: no, bf: true, ag: 8)
```

1.3 Copy a LV with cplv:

The cplv command:

Since a LV corresponds to a filesystem (if one is created on that LV),
you can copy a LV (or thus a Filesystem) from one VG to another VG (or in the same VG).

Syntax:

-- Copying to a new LV called "NeWLogicalVolume"
```
cplv -v VolumeGroup -y NewLogicalVolume SourceLogicalVolume
```

-- Copying to an already existing LV named "DestinationLogicalVolume"
```
cplv -e DestinationLogicalVolume SourceLogicalVolume
```

With the "-e" flag, you can specify that the DestinationLogicalVolume already exists (with that name).

If you specify the -v VGName, then the new LV will be created in VGName.
Otherwise, the new LV will be created in the same VG.

If you specify -y NewLogicalVolume, the new LV takes that name, otherwise the system will generate one.
cplv will do an exact PP copy from the source to the destination.
It's an alternative to copy filesystems with shell commands.

Examples:

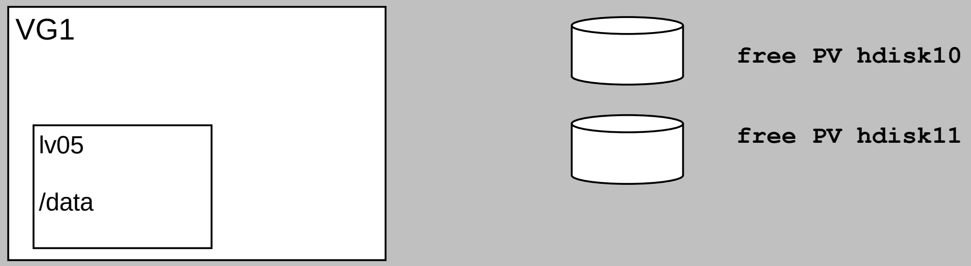
```
# cplv lv09                  # will create a new LV, as a copy of lv09,  with a system generated name, in the same VG.
# cplv -v vg2 lv09           # will create a copy of lv09 in Volume Group vg2.
```

The next command will create a new LV called "lvnew" as a copy of "lvold" in VG "vgnew":

```
# cplv -v vgnew -y lvnew lvold                # Take notice of this command !
```

An example of a move of a "live" filesystem:

Suppose we have the following situation:



Right now, we have a VG called "vg1", containing the LV called "lv05".
On lv05, the filesystem "/data" was created. Suppose that by now "/data" contains 20 GB data of users.
Suppose furher that we need to move "/data" to a new Volume Group "vg2" (suppose vg1 get's too full).

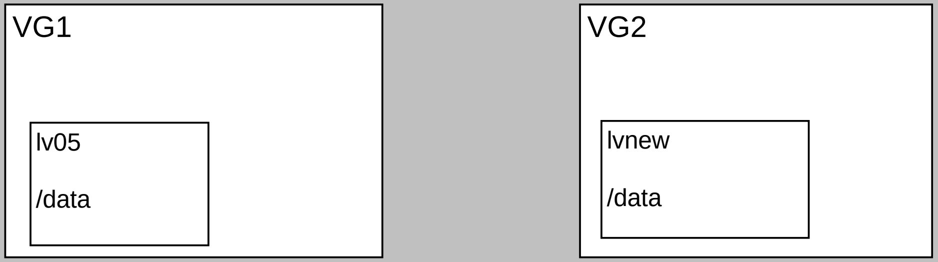
- Step 1: Create the Volume Group vg2

```
# mkvg -y vg2 -s 128 hdisk10 hdisk11
```

- Step 2: Create a new LV on vg2:

```
# mklv -y lvnew -c 2 vg2 200         # the new LV is called lvnew with a size of 200 x 128= 25.6 GB
```

Actually, we are not required to create a new LV. The cplv command can create one for us.
But, since we want mirroring in effect in the new LV (and possibly other options as well),
we create one ourselves.



Step 3: umount the "/data" filesystem, so that it is closed and no user access is possible.

```
# umount /data
```

Step 4: optionally rename the filesystem.

```
# chfs -m /data_old /data
```

You are not required to do this. But in this case, we want the new filesystem to use the former mountpoint
of "/data". So, with the "chfs" command we change the mountpoint from "/data" to "/data_old".

Step 5. Now cplv "lv05" to "lvnew":

```
# cplv -e lvnew lv05                 # we use the -e flag, because our new LV already exists.
```

Note:
If you see this error:

```
0516-746 cplv: Destination logical volume must have type set to copy.
```

Then use the following command:

```
# chlv -t copy lvnew
```

Now it's copying all the information of the PP's of lv05 to lvnew.
In this example, lv05 contained 20 GB worth of data, so it probably will take some time.

After the copying is done, we need to do some housekeeping:
We need to make sure that the "/data" filesystem is associated to the new LV "lvnew".

```
# chfs -a dev=/dev/lvnew /data_old
# chfs -m /data /data_old

# mount /data
```

We are done copying.
After you are sure everyting is OK on the new copy, we can optionally remove lv05:

```
# rmlv lv05       # in order to free space in vg1.
```

Or, what is similar to the rmlv command:

```
# rmfs /dev/lv01

Warning, all data contained on logical volume lv05 will be destroyed.
rmlv: Do you wish to continue? y(es) n(o)? y
rmlv: Logical volume lv05 is removed.
```

Alternative for the 'cplv" command:

As an alternative to the cplv command, if you want to copy filesystems,
from the 'one location' 'to another location', you can consider using cpio.
This command will be discussed in another chapter about backup & recovery.
But suppose you have a source dir "/data/dira", and you want to copy all files and subdirectories,
including the filemodes (permissions) and ownership, to, say "/data2/dirb", then use cpio
as in the following example:

```
# cd /data/dira
# find . | cpio -pdvm /data2/dirb
```

"/data/dirb" will be an exact copy of "/data/dira", including all subdirectories, permissions, etc..

Don't you agree that the above command is really great?

1.4 Import and Export of a Volume Group.

This short section deals with the "export" (using exportvg) and "import" (using importvg)
of a volume group.

Any PV in a VG has at least one copy of the VGDA (and other special area's). This makes a VG "selfdescribing".
This also makes a VG, in principle, transportable.
But exportvg and importvg are also used to solve incorrect ODM entries.

Let's take a look at this very simple example:

In the following example, the VG is called "datavg" consisting of the PV hdisk5.

Unmount all filesystems in that VG first, otherwise you cannot varyoff the VG.
Then varyoff the VG.

```
# varyoffvg datavg
```

Now remove the complete information of that VG from ODM. The VGDA and LVCB
on the actual disks are NOT touched by the exportvg command.

```
# exportvg datavg
```

Now import the VG and create new ODM objects associated with that VG:

```
# importvg -y datavg hdisk5
```

You only need to specify one intact PV of the VG in the above command. Any disk in the VG

will have a VGDA which contains all neccessary information.
The importvg command reads the VGDA and LVCB on that disk and creates completely new ODM entries.

1.5 Some other important facts:

1.5.1. File systems that are located in ROOTVG:

Let's see what LV's and filesystems are stored on the ROOTVG:

Example: listing of logical volumes on rootvg:

```
# lsvg -l rootvg
LV NAME       TYPE         LPs    PPs    PVs     LV STATE       MOUNT POINT
hd5           boot         1      1      1       closed/syncd   N/A
hd6           paging       24     24     1       open/syncd     N/A
hd8           jfslog       1      1      1       open/syncd     N/A
hd4           jfs          4      4      1       open/synced    /
hd2           jfs          76     76     1       open/synced    /usr
hd9var        jfs          4      4      1       open/synced    /var
hd3           jfs          6      6      1       open/synced    /tmp
paging00      paging       20     20     1       open/synced    N/A
```

If you forgot what "lsvg -l <VG_Name> does for you, then review section 1.1.5.

Memorize the following table with the default LV's and mountpoints:

LV name	mountpoint
hd4	/
hd5	boot logical volume
hd6	paging
hd2	/usr
hd3	/tmp
hd1	/home
hd9var	/var
hd10opt	/opt
hd8	jsflog
hd11admin	/admin contains /admin/tmp on AIX 6.1

1.5.2. Types of Volume Groups:

Type:	Version	Max Physical Volumes	Max Logical Volumes
Normal VG	AIX5	32	256
Big VG	AIX5	128	512
Scalable VG	as of AIX5.3	1024	4096

1.6 Some pointers on commands in LVM troubleshooting:

In this section, we will show some short examples in using a few special commands, used with LVM problems.

ATTENTION: this section is NOT about troubleshooting LVM problems.
I just want you to know the main purpose of some of those commands, and that's really all.
Ofcourse, if, in reality, you suspect serious problems in LVM, you really need much better information
than what you can find in this section. Remember, this document is only a supporting document on Exam 223.
But as the last chapter of this document we will review some troubleshooting and diagnosing in general,
and review a few cases in LVM troubleshooting

All disks, or PV's, that are member of a VG, have a couple of special area's at the beginning of the disk.
We already have discussed the VGDA in a former section.

Here is a short description of those special area's:

- 1. VOLUME GROUP DESCRIPTOR AREA, VGDA

Global to the VG:
The VGDA, located at the beginning of each physical volume, contains information that describes all
the LV's and all the PV's that belong to the VG of which that PV is a member.
The VGDA makes a VG selfdescribing. An AIX System can read the VGDA on a disk, and from that, can
determine what PV's and LV's are part of this VG.
There are one or two copies per disk.

- 2. VOLUME GROUP STATUS AREA, VGSA

The VGSA contains state information about physical partitions and physical volumes.
For example, the VGSA knows if one PV in a VG is unavailable. For example, you might
have shut down a PV with the "chpv" command.
Each PV has at least one VGDA/VGSA. The number of VGDA's contained on a single disk
varies according to the number of disks in the VG.

- 3. LOGICAL VOLUME CONTROL BLOCK, LVCB

Contains LV attributes (like number of copies).
The LVCB is located at the start of every LV. It contains information about the logical volume.
You can however, use the mklv command with the -T option, to request that the LVCB will not
be stored in the beginning of the LV.
With Scalable VG's, LVCM info is no longer stored in the first user block of any LV.
All relevant LVCM info is kept in the VGDA.

With the "getlvcb" command, you can see those attributes in a good readable format.

Let's spend a few words (but not enough words !) on the following commands (so that you alread know about them):

```
lqueryvg
lquerypv
ldeletepv
chpv
synclvodm
getlvcb
```

The lqueryvg command:

The lqueryvg command reads the VGDA from a specified disk in a VG.

Most important Flags:
 -p: which PV
 -A: show all available information
 -t: show descriptive tags

Example:

```
# lqueryvg -Atp hdisk0

Max LVs:        256
PP Size:        25
Free PPs:       468
LV count:       20
PV count:       2
Total VGDAs:    3
Conc Allowed:   0
MAX PPs per PV  1016
MAX PVs:        32
Conc Autovaryo  0
Varied on Conc  0
Logical:        00c665ed00004c0000000112b7408848.1   hd5 1
                00c665ed00004c0000000112b7408848.2   hd6 1
                00c665ed00004c0000000112b7408848.3   hd8 1
                00c665ed00004c0000000112b7408848.4   hd4 1
                00c665ed00004c0000000112b7408848.5   hd2 1
                00c665ed00004c0000000112b7408848.6   hd9var 1
                00c665ed00004c0000000112b7408848.7   hd3 1
                00c665ed00004c0000000112b7408848.8   hd1 1
                00c665ed00004c0000000112b7408848.9   hd10opt 1
                00c665ed00004c0000000112b7408848.10  hd7 1
etc..
 (many rows omitted)
```

The lquerypv command:

Used to get a dump from the special area's on a PV.

lquerypv -h /dev/PVname | -p PVid -g VGid | -N PVname offset length
The below command shows the PVID that is stored on hdisk5, which can be found on offset 80

```
# lquerypv -h /dev/hdisk5 80 10
00000080   00001155 583CD4B0 00000000 00000000  |...UX<.........|
```

The ldeletepv command:

To remove a "phantom disk" from the ODM, use reducevg with the PVID instead of the disk name.
But sometimes even that does not work.
To clean ODM, you might consider using the ldeletepv command:

```
# ldeletepv -g VGID -p PVID
```

When you are at the stage that you consider using the ldeletepv command, surely you have seen already some error message, stating the involved VGID and PVID.

The chpv command:

The chpv command changes the state of the physical volume in a volume group.
Just like the examples above, it's mainly used in troubleshooting LVM problems, and I just want you to know a little about this command.

Examples

To close physical volume hdisk03, so that all IO will prevented,enter:

```
# chpv -v r hdisk03
```

To open physical volume hdisk03 again for all IO, enter:

```
# chpv -v a hdisk03
```

Any PV has a bootrecord, and in some cases it needs to be cleared.
So, in some troubleshooting situations, if you need to clear the boot record of for example hdisk3, enter:

```
# chpv -c hdisk3
```

The synclvodm command:

In rare occasions, the LVM information in the ODM, is not in sync anymore with the VGDA's in a Volume Group. It is possible to correct that with the "synclvodm" command.
There are a number of indicators that shows that there are indeed inconsistencies, like e.g.

```
0516-306 getlvodm: Unable to find lv09 in the Device
Configuration Database.
```

So here AIX complains that lv09 is not found in ODM. Maybe, while a sysadmin was trying to create it, accidently pressed Ctrl-C (or something), while he/she thought he/she was busy in another ssh session.

The synclvodm command, will read the VGDA from the VG specified and updates the ODM database.
This command is typically used to fix inconsistencies in the ODM database.

Suppose in the above errormessage, you know that the VG involved is "appsvg", you mighty try this:

```
# synclvodm -v appsvg
```

The getlvcb command:

As you may have read above, the LVCB stores attributes of a LV.
The getlvcb command reads the LVCB of a specified LV, and presents it in a nice formatted form.

```
# getlvcb -AT lv05

         AIX LVCB
         intrapolicy = m
         copies = 2
         interpolicy = m
         lvid = 00c8beec00005c00000001049820e38f.17
         lvname = lv05
         machine id = 6BCCC4C00
         number lps = 32
         relocatable = y
         strict = y
         stripe width = 0
         stripe size in exponent = 0
         type = jfs2
etc..
```

Using "exportvg" and "importvg" in case of ODM problems:

From section 4.4, you know that the exportvg command, does not touch the VGDA on the Physical Volumes. *It only clears the ODM with respect to this particular VG*. Indeed, the command was meant "to export" a VG to "somewhere", so the ODM must be cleared.

So if you export, and thereafter import a VG, you essentially "renew" the ODM information. This is so, because the "importvg" command, reads the VGDA (of a specified disk) and updates the ODM accordingly.

Example:

Unmount all filesystems in that VG first, otherwise you cannot varyoff the VG. Then varyoff the VG.

```
# varyoffvg myvg
```

Now remove the complete information of that VG from ODM. The VGDA and LVCB on the actual disks are NOT touched by the exportvg command.

```
# exportvg myvg
```

Next, we import the VG with the importvg command. You can use any PV from that VG, because they all have at least one VGDA

```
# importvg -y myvg hdisk3
```

1.7 Some other commands and concepts related to LVM and fileystem management:

In this chapter, we have seen a number of important LVM related commands, like:

What we have seen sofar:	
listing objects (PV, VG, LV):	lspv lsvg lslv
creating objects (VG, LV):	mkvg mklv
creating filesystems:	crfs mkfs
listing and changing filesystems:	lsfs chfs
removing a LV (and filesystem)	rmlv
extending a LV (adding PP's)	extendlv
copy a logical volume	cplv
remove a disk from a VG: *add a disk to a VG:*	reducevg extendvg
exporting and importing a VG:	exportvg importvg

And, we have seen some commands that can help in troubleshooting LVM problems,
or retreiving special information, like "lqueryvg".

We still need to adress a couple of other important commands and concepts that relate to LVM objects.
Commands and concepts that are related to *backup and performance,* will be covered in other chapters.

lvmstat:

Reports input/output statistics for logical partitions, logical volumes and volume groups.
It is specifically meant to show the IO on the LV's of a VG, or of all Partitions in a LV.

```
lvmstat { -l | -v } Name [ -e | -d ] [ -F ] [ -C ] [ -c Count ] [ -s ] [ Interval [ Iterations ] ]
```

By default, the statistics collection is not enabled in the system. You must use the -e flag to
enable this feature for the logical volume or volume group in question. Enabling the statistics collection
for a volume group enables for all the logical volume in that volume group.

The first report generated by lvmstat provides statistics concerning the time since the system was booted.
Each subsequent report covers the time since the previous report. All statistics are reported each time lvmstat runs.

>> To enable/disable the statistics for a specific logical volume, use the following command:

```
# lvmstat -l lvname -e                # enable
# lvmstat -l lvname -d                # disable
```

>> To enable/disable the statistics for all logical volumes in a volume group, use the following command:

```
# lvmstat -v vgname -e                # enable
# lvmstat -v vgname -d                # disable
```

>> To display the history of all the partitions of logical volume hd2, type:

```
# lvmstat -l hd2
```

>> To display the history of top five logical volumes of volume group uservg, type:

```
# lvmstat -v uservg -c 5
```

>> To display six reports at two second intervals for the volume group rootvg, type:

```
# lvmstat -v rootvg 2 6
```

Example:

```
# lvmstat -v rootvg

Logical Volume       iocnt    KB_read   KB_wrtn   Kbps
hd8                   4        0         0         0.00
paging01              0        0         0         0.00
..
```

Quorum:

A Volume Group can consist of multiple Physical Volumes (disks). It could happen, that a Physical Volume malfunctions for some reason. Whether the whole Volume Group can stay online, depends on the "quorum" as it is at that time.
As you know, each Physical Volume has a VGDA area.

The quorum is the minimum number of VGDAs, or Physical Volumes, that must be good to keep the VG online.
As you will understand, the quorum is thus a critical number.
In AIX, per default the following situation is in effect:

The quorum by default must be greater than 51% of the total of VGDAs.
The VGDAs are divided in the Volume Group on the Physical Volumes according to the amount of physical disks:

VG with one PV:	The disk contains two VGDAs.
VG with two PVs:	The first disk contains two VGDAs and the second disk contains only one VGDA.
VG with 3 or more PVs:	Each disk contains only one VGDA.

Per defdault, if a VG "looses" it's chorum, it will be varied off. So if the number of Physical Volumes that are "up", is below the quorum, the VG will also not activate when you try to vary it on.

You can use the chvg command to change that behaviour.

chvg command:
The chvg command changes the characteristics of a volume group.

```
-Q flag:          Determines if the volume group is automatically varied off after losing its quorum of physical volumes.
                  The default value is yes. The change becomes effective immediately.
                  n: Will stay active until it looses all Physical Volumes.
                  y: the Volume Group is automatically varied off if it looses it's quorum.
```

```
# chvg -Qn vg02
```

Chapter 2. Some keypoints about Tape devices.

2.1 About Tape Devices:

Tapedevices on AIX are named like "/dev/rmt*", like /dev/rmt0 (your first device) and /dev/rmt1 (second device) etc...
But such corresponding hardware, has multiple "instances" (special devices), and all generally have different properties.
Take a look at the following table:

Special_File	Rewind_on_Close
/dev/rmt*	Yes
/dev/rmt*.1	No
/dev/rmt*.2	Yes
/dev/rmt*.3	No
/dev/rmt*.4	Yes
/dev/rmt*.5	No
/dev/rmt*.6	Yes
/dev/rmt*.7	No

So, you might create simple backups, using rmt0 or rmt1, like:

```
tar -cvf /dev/rmt0 /appl             # tape archive (tar) backup of the /appl filesystem to tape mounted on rmt0
or
tar -cvf /dev/rmt1 /appl             # tape archive backup of the /appl filesystem to tape mounted on rmt1
```

Remark: backup commands like "backup", "tar", "cpio" etc.., are covered in another chapter.

But, if you use a script, using for example series of tar commands, the special file you use is very important!
That is so, because for example "rmt0" will rewind after close (after each tar command has ended)
while "rmt0.1" does not rewind !

Using (for example) rmt1 or rmt1.1?

So, if your script uses a series of commands to backup a listing of filesystems, then you better use rmt0.1 or rmt1.1 etc..
Those special devices does not rewind after each command, and thus will not overwrite any previous content
that was on the tape, from the previous command.

This script then is OK:

```
tar -cvf /dev/rmt1.1 /appl
tar -cvf /dev/rmt1.1 /data
tar -cvf /dev/rmt1.1 /oradata/dumps
```

Here you have three "files" on tape,
representing your three backups.

This script then is WRONG:

```
tar -cvf /dev/rmt1 /appl
tar -cvf /dev/rmt1 /data
tar -cvf /dev/rmt1 /oradata/dumps
```

Here, the only backup you have is the last one: "/oradata/dumps"
Each command, overwrites the result of the former command.

Note: the point here is to stress the importance of the non-rewinding class of a tape device.
If you just take into account usable flags of the tar command (or other archiving command), then tar itself
will be able to manage things for you. Take a look at those tar flags:

```
-c create         Used to create a new tape archive (tar) file.
-r append         So, with "append" you can just "add" backups, instead of creating new "tape archives".
-x extract
-v verbose
-t list
```

As noted before: backup commands will be discussed in another chapter.

If you use as the first tar command "tar -cvf", and as the following tar commands "tar -rvf"
you will just append into the archive, instead of creating new archives.
So, deciding on "-c" and "-r" in using tar is important. The "-v" flag is just a "verbose" flag.

Listing product data from the device:

As we have already seen in chapter 1, producing product data from your device can be done with the "lscfg" command:

```
# lscfg -vl rmt0
Manufacturer...............EXABYTE
Machine Type and Model.....IBM-20GB
Device Specific(Z1)........38zA
Serial Number..............60089837
..
```

Listing Availability of tape device(s):

Also, as we have see from chapter 1, we can view if a device is actually configured, or if
it has only the "Defined" status in the ODM:

```
# lsdev -Cc tape
rmt0  Available  10-60-00-5,0  SCSI 8mm Tape Drive
```

2.2 The "mt" or "tctl", and "tapeutil" commands:

2.1.1 The "mt" or "tctl" commands:

The "magnetic tape" mt command (or "tctl" command) is used to control your mounted tapes.
Normally, you would use the "-f" parameter, to designate which tapedevice you want to control.
For example, you might have both "/dev/rmt0" and "/dev/rmt1", and by using -f you can tell "mt" which
device you want it to operate on.
If you leave out the "-f", then if the TAPE variable is in place, then that device will be taken.

The syntax of mt is:

`mt -f <devicename> <subcommand>`   like for example

```
# mt -f /dev/rmt0 rewind
```

The following important subcommands can be used:

rewind, rewoff1	rewinds the tape
fsf no_of_files	fast skip forward: Moves the tape forward the number of files (or "markers", or "blocks") specified by the "no_of_files" parameter and positions it to the beginning of the next file.
bsf no_of_files	back skip forward: Moves the tape backwards the number of files specified by the no_of_files parameter and positions it to the beginning of the last file skipped.

Example of usage:

Suppose we use the following script to backup some filesystems.

```
                                    # Comment                                   "logical tapepointer or pos. at file#"
mt -f /dev/rmt1 rewind              # first rewind the tape
tar -cf /dev/rmt1.1 /apps           # will be the 1st backupfile on tape                     0
tar -cf /dev/rmt1.1 /software       # will be the 2nd backupfile on tape                     1
tar -cf /dev/rmt1.1 /opt            # will be the 3rd backupfile on tape                     2
tar -cf /dev/rmt1.1 /var            # will be the 4th backupfile on tape                     3
tar -cf /dev/rmt1.1 /u01/oradata    # will be the 5th backupfile on tape                     4
tar -cf /dev/rmt1.1 /u02/oradata    # will be the 6th backupfile on tape                     5
tar -cf /dev/rmt1.1 /u03/oradata    # will be the 7th backupfile on tape                     6
```

Now, suppose at a later time, the filesystem "/u01/oradata" gets corrupted, and
after rebuilding that filesystem, we want to restore it from our latest tape.
We can do that in the following way:

1. First, rewind the tape:

```
# mt -f /dev/rmt1 rewind
```

2. Move the tape 4 files ahead, so that it is positioned at the beginning of the backup of "/u01/oradata".
   If you move 4 files, then you are at the beginning of file# 5, which is the beginning of "/u01/oradata".

```
# mt -f /dev/rmt1.1 fsf 4
```

3. Now, restore the data of "/u01/oradata":

```
# tar -xvf /dev/rmt1.1 /u01/oradata                     # note: to restore with tar, use the -x flag
```

The "tctl" command can be used much in the same way as the "mt" command, in managing your tapedevice.
It even has more subcommands compared to "mt".

Examples:

To rewind the rmt1 tape device, enter:

```
# tctl  -f /dev/rmt1 rewind
```

To move the tape to the third file (or "marker" or "block"):

```
# tctl -f /dev/rmt0.1 fsf 3
```

And if the TAPE variable is set (or if /dev/rmt0.1 is the default), then just use:

```
# tctl fsf 3
```

2.2.2 the tapeutil command:

If you have installed a "tape library" (any sort of robot with a magazine, either be small or large),
then you can use the "tapeutil" command to program the library.

On AIX, typically, your first tape library would be named "smc0".

For example, suppose you have a small pSeries machine, with only the local rmt0 device.
After installing a certain tapelibrary, you would use the "lsdev" command like so:

```
 # lsdev -Cc tape
rmt0 Available 08-08-00-0,0 LVD SCSI 4mm Tape Drive
rmt1 Available 06-08-00-2,0 IBM 3580 Ultrium Tape Drive
smc0 Available 06-08-00-2,1 IBM 3576 Library Medium Changer
```

In this case, the new tape device is "/dev/rmt1", which uses the interface / library "smc0".

Now, this device could have only one "drive" (/dev/rmt1) and a magazine with a number of "slots", containing tapes.
You then could use the tapeutil command to load a tape from a certain slot into the drive, and the other way around.
/usr/bin/tapeutil -f /dev/smc0 move <fromslot> <toslot>
For example:

```
# tapeutil -f /dev/smc0 move -s 10 -d 23
```

This upper command, moves the tape from slotnumber 10, into the tapedrive designated by 'd 23'.

For the exact details of such a command (of a certain real library), you should refer
to the manual of that particular library.

2.3 Some other commands, and examples, related to tapedevice:

chdev for changing device properties as block_size:

The "chdev" (change device) command, is generally used to change properties of a device (like an ethernetcard,
or tapedevice). In it's simplest for, this command has the syntax:
chdev -l <device> -a <property=value>

So, suppose that you want to change the "block_size" property of your drive, and set it to value of 512 bytes, use

```
# chdev -l /dev/rmt0 -a block_size=512
```

Some backup command examples:

Next, lets view a couple of representative commands on how to make backups to a tape mounted on rmt0.
All those backup commands will be discussed in a later chapter, but it might be interresting to view how
they look like:

```
# mksysb -i /dev/rmt0               # makes an Operating System backup with rootvg to tape.
# backup -0 -uf /dev/rmt0 /data     # backup the "/data" filesystem, using the backup command, to tape.
# savevg -if /dev/rmt0 uservg       # backup the volume group "uservg" to tape, using the "savevg" command.
# tar -cvf /dev/rmt0 /data          # backup the "/data" filesystem, to tape, using the tar command.
```

The "tcopy" command:

The tcopy command copies magnetic tapes, like "tcopy source_tape destination_tape"
If you only specify the source, the tcopy command prints information about the size of records and tape files.

Example:

```
# tcopy /dev/rmt0 /dev/rmt1
```

Note: you could also use the dd command, like this:

```
# dd  if=/dev/rmt0  ibs=1024  obs=1024  of=/dev/rmt1
```

Chapter 3. Some keypoints on Memory.

There are many new things in AIX v6.1, compared to AIX v5.3, *but with respect to memory management*, *the differences are really not so terribly large.* Rather, the difference in memory management between for example v5.1 to v5.3 are more profound than going from v5.3 to v.6.1.
The commands to view and manage memory (like "vmo") and the common monitoring tools have not changed from v5.3 to v6.1.
But, some important vmo *default values*, *did* changed though.

There is certainly a lot "to understand" when pondering about memory in AIX, but at the same time, there are also a lot of facts that are "just the way they are".

Let us just walk through *a number of* important aspects of memory.

3.1. Pages and segments:

One important aspect of AIX memory is VMM: The role of Virtual Memory Manager (VMM) is to provide the capability for programs to address more memory locations than are actually available in physical memory.
On AIX this is accomplished using segments that are partitioned into fixed sizes called “pages”.

- A memory segment is 256M
  This is true for the older 32bit memory model, and is still true for the 64bit model.
  A segment can be "shared" (accessible by multiple processes), or "private".
- the default page size is 4K
- POWER 4+/5/6 can define large pages, which are 16M
- As of Power5+, you can also use 64KB (as of AIX 5.3 5300-04), and 16GB pages.

Such a page, is the unit of IO with respect to paging. So, although it's the default, 4K seems a bit small,if many pages will swap in and out. At the same time, a unit of 16GB seems only appropriate for very heavy workloads on very large memory (theoretically).

On systems that support 64KB pages, the AIX kernel will automatically configure 64KB pages for the system. 64KB pages are fully pageable, and the size of the pool of 64KB page frames on a system is dynamic and fully managed by AIX.
Pools of 4K an 64KB pageframes can exists at the same time on a system.

The 16M and 16GB page sizes are intended to be used only in very high-performance environments, and AIX will not automatically configure a system to use these page sizes. A system administrator must configure AIX to use these page sizes and configure the number of pages of each of these page sizes. Furthermore, AIX will not automatically change the number of configured 16MB or 16GB pages based on demand.

Although tuning is not exclusively done by the vmo command, it's a most widely used tool.
With respect to pagesizes, you often have to revert to vmo, as in the following example:

```
# vmo -r -o lgpg_regions=64 -o lgpg_size=16777216                # here, 1GB of 16M pages are configured.
```

Whether you need to reboot depends if DLPAR features are available on your platform, and have been implemented.
Configuring 16GB page support, and poolsize, must be done using the HMC.

Pages of different size said to be in different "memory pools". So, while only using 4K and 16M pages, you might end up with for example 5 pools at a certain time.

It should be noted that on AIX v6, many vmo settings are considered to be restricted, which means that you should check with Support if your change is a viable option.
But you can do such a change anyway.

3.2. 32bit and 64 bit models:

3.2.1 32 bit models: The Small Memory Model and Large memory Model:

This sections deals with 32 bit applications only.
How many segments a program can use really depends on the memory model used, and if the app is 32 or 64 bit.
Here we will talk solely about 32bit programs in a 32bit environment.
Don't forget: Virtual memory is at work, so each program has a 4GB address space.

The 4GB memory space (per program (!); remember, it's virtual) is divided into 16 x 256 segements.

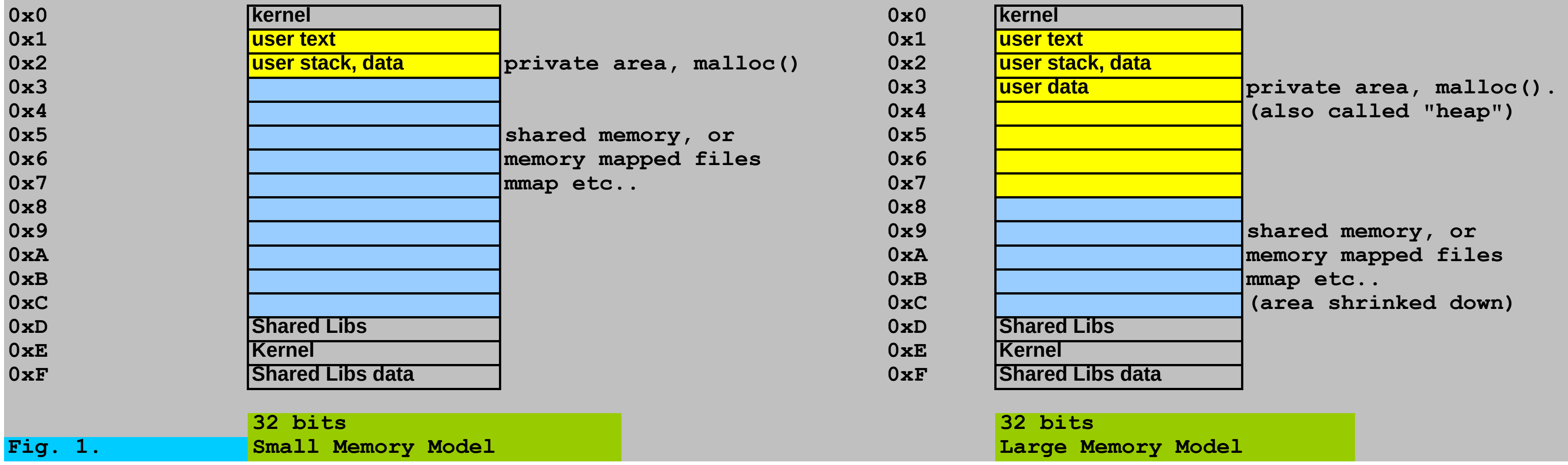


Fig. 1.

using
LDR_CTRL=MAXDATA=N segments
or compiling with bmaxdata=N segments

32bit: The Small Memory Model:
Here, AIX assigns a virtual address space partitioned into 16 segments of 256 MB.
In the 'Small Memory Model' for an application, the application has only one segment, segment 2 (0x2), in which it can malloc() data and allocate additional thread stacks. This stack area is thus a private area.
It does, however, have 11 segments of shared memory into which it can mmap() or shmap() data.

32bit: The Large memory Model:
You can control the boundary between Private and Shared memory, giving more Private memory to the application, and at the same time reducing the amount of Shared memory.

You can devide memory into "private" memory area's (private for an application), and shared memory, which can be accessed by multiple processes, or just for memory-mapped files.
By using mmap like functions, the process can also map files into that shared memory.
It's not uncommon, to use an environment variable in the profile of the credentials under which an application start, to set aside more private memory (at the cost of shared memory).
The following is an example for a 32bit application:

```
LDR_CNTRL=MAXDATA=0x50000000
```

which means that 4+1=5 segments are reserved for that program's private working area.
Especially using 32bit JVM, these methods were widely used, to expand the native heap.
For programs under your control (having sources), you could also use the "bmaxdata" compiler option, like so:

```
# cc test.o -bmaxdata:0x60000000
```

The number 0x60000000 is the number of bytes, in hexadecimal format, which is 6 x 256 MB segments

32bit: The Very Large memory Model:

Actually, there is a third option as well. This was called the "Very large Memory Model".
In this case, programs had to be compiled using "-bmaxdata:0xN0000000 DSA"
where DSA is "Dynamic Segment Allocation". This allowed for 8 private segments, allocated in a dynamic way.

Remark about the "EXTSHM" environment variable:
In the 32bit architecture, a process is per default allowed to attach to a maximum of 11 shared memory segments .
Using EXTSHM, *really* changes the nature of the shared memory segments that are attached by an application .
The "granularity" of the "segment-attach" then changes fundamentally, so that it does not involve a complete segement anymore. Practically speaking, the allocation is on the "n x Page" level.
This will prevent the "*segment table full*" errors, but some users reported some performance issues.

(Although these are all 32bit issues, at the AIX 53 platform, fixes became available that enabled this variable for 64bit programs. This was done to improve the interoperability between 32 and 64 bit programs)

In a sense you might say: this is all AIX 4.2 stuff, since 64bit support already was available in 4.3.
Not really. So many apps remained 32bit for such a long time, and some even still are today.

3.2.2 The 64 bit memory model:

This virtual memory is *incredebly* much larger compared to the 32bit model.
256 MB memory segments are still used, but about 4,300,000,000 segments are available instead of 16.
Segments are dynamically allocated within specified memory ranges for various uses, like private memory
or shared memory. User data, or private area, can now occupy up to 448 Petabytes of memory.

3.3. "Computational" and "Non Computational" Memory.

Most performance monitoring tools, show items that might easily escape your attention.
For example, on AIX, the "topas" utility (used on a lpar) is highly appreciated. Even if you only use it
to get a "quick" "real_time" view on performance. For example, you want to know how busy the cpu's are,
the current memory occupation, how the paging space is used etc..
Note: for people no so aquinted with AIX, "topas" looks a lot like "nmon", found on most other platforms.

Here is an example screen of topas:

```
# topas
```

Fig. 2

```
Topas Monitor for host:    stargate              EVENTS/QUEUES    FILE/TTY
Mon Dec 14 16:16:50 2009   Interval:  2          Cswitch     5984 Readch     4864
                                                 Syscall    15776 Writech   34280
Kernel   63.1   |##################          |   Reads          8 Rawin         0
User     36.8   |##########                  |   Writes      2469 Ttyout        0
Wait      0.0   |                            |   Forks          0 Igets         0
Idle      0.0   |                            |   Execs          0 Namei         4
                                                 Runqueue    11.5 Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out   Waitqueue    0.0
lo0     213.9   2154.2  2153.7   107.0   106.9
tr0      34.7     16.9    34.4     0.9    33.8   PAGING           MEMORY
                                                 Faults      3862 Real,MB    1023
Disk    Busy%     KBPS     TPS KB-Read KB-Writ   Steals      1580 % Comp     27.0
hdisk0    0.0      0.0     0.0     0.0     0.0   PgspIn         0 % Noncomp  73.9
                                                 PgspOut        0 % Client    0.5
Name          PID CPU% PgSp Owner                PageIn         0
java        16684 83.6 35.1 root                 PageOut        0 PAGING SPACE
java        12192 12.7 86.2 root                 Sios           0 Size,MB     512
lrud         1032  2.7  0.0 root                                  % Used      1.2
aixterm     19502  0.5  0.7 root                 NFS (calls/sec)  % Free     98.7
topas        6908  0.5  0.8 root                 ServerV2       0
ksh         18148  0.0  0.7 root                 ClientV2       0   Press:
gil          1806  0.0  0.0 root                 ServerV3       0   "h" for help
```

This time, I would like your attention focused on the small block in blue: "%Comp" and "%Noncomp".

%Comp: Computational Memory:
Simply said: It's the percentage of real memory used by all processes.
Computational pages are pages used for the text, data, stack, and shared memory of a process

%Non Comp:Non Computational Memory:
Simply said: It's the percentage of real memory used to cache data files.

%Client:
These are the percentage of cached pages, due to remote data like NFS.

So in the example above, the "filecache" is larger than what all processes combined uses from the memory.
You cannot draw conclusions from this allone. You also need to know the purpose of this specific lpar (machine).

Suppose the machine works mainly like a "File Server", then the filecache should indeed be large.
But suppose it's a database machine, then there might be quite a lot of private user session area's (processes),
as well as large shared memory segments that should be left to handle by the RDBMS('ses) themselves.

That's why in this section, I attended you to the Computational or Non Computational memory percentages.
Maybe, for a certain role for your machine, the one should favor the other.

But if the machine does not have a specific database role (or something similar) then per default
AIX displays an quite "hungry" filecaching algolrithm. So, if the machine is up for some time, it
generally will "appear" "*as if*" your free memory is low.
Indeed, then it actually is. But pages will be replaced by "wanted" pages from disk or pagefile,
if page faults occur.
But you have quite some control on how the system will replace filecache pages or computational pages.

3.4. Some Commands showing Memory or memory utilization:

3.4.1. The "lparstat -i" command:

As of AIX5.3, this command is available.
The information that is displayed, is actually in the context of "virtualization", so you can view
for example how many "virtual cpu's" are assigned to your lpar, and many other aspects that are related to virtualization.

With respect to memory, you can see how much Memory is currently assigned to the lpar, and what the
minimum an maximum values are to where your memory can shrink or grow to.

```
# lparstat -i

Node Name                     : stargate5
Partition Name                : stargate5
..
..
Online Memory                 : 4096
Maximum Memory                : 8192
Minimum Memory                : 2048
..
etc..
```

3.4.2. Some quick commands that will show you the total real memory:

```
# bootinfo -r
# lsattr -E -l mem0
# lsattr -E -l sys0 -a realmem
# prtconf | grep Mem
```

3.4.3. A nice summary on VMM using "vmstat -v":

On all unixes, "vmstat" is used to show statistics on virtual memory usage, paging status, and cpu usage.
On AIX, if you call vmstat using the "-v" flag, "vmstat -v", you get a comprehensive overview on
various statistics maintained by the Virtual Memory Manager.
Example:

```
# vmstat -v
                  4030464 memory pages
                  3822276 lruable pages
                  2565 free pages
                  1 memory pools
                  794538 pinned pages
                  80.0 maxpin percentage
                  20.0 minperm percentage
                  80.0 maxperm percentage
                  67.4 numperm percentage
                  2578020 file pages
                  0.0 compressed percentage
                  0 compressed pages
                  0.1 numclient percentage
                  80.0 maxclient percentage
                  7588 client pages
                  0 remote pageouts scheduled
                  3985 pending disk I/Os blocked with no pbuf
                  750525 paging space I/Os blocked with no psbuf
                  11983944 filesystem I/Os blocked with no fsbuf
                  0 client filesystem I/Os blocked with no fsbuf
                  3937 external pager filesystem I/Os blocked with no fsbuf
                  0 Virtualized Partition Memory Page Faults
                  0.00 Time resolving virtualized partition memory page faults
```

Some of the most important output records (for now) are:

file pages	Number of 4 KB pages currently used by the file cache.
lruable pages	Number of 4 KB pages considered as possible candidates for replacement (lru least recently used).
free pages	Number of free 4 KB pages.
memory pages	Size of real memory in number of 4 KB pages.
numclient percentage	Percentage of memory occupied by client pages.
client pages	Number of client pages (usually due to caching remote files).
numperm percentage	Percentage of memory currently used by the file cache.
maxperm percentage	This specifies the point above which the page stealing algorithm steals only file pages.
minperm percentage	This specifies the point below which file pages are protected from the re-page algorithm.
maxclient percentage	This specifies the maximum of the client cache due to cached pages from remote files and JFS2.

Note: On AIX v6, vmstat can show us statistics on WPAR's as well. This will be a subject in another chapter.

3.4.4. Using the vmo command with "-L" as a reporting flag:

The "vmo" is used to tune (change) many memory "tunables".
But if you want to see the values what they are now, use the vmo command with the "-L" flag.

Examples:

The following command produces a large output. All vmo tunables are shown with their current values.

```
# vmo -L
```

The following command shows the min and max values controlling file cache:

```
# vmo -L minperm% -L maxperm%
```

3.5. A few examples on tuning VMM using vmo:

There are many tunables that can be controlled by the "vmo" command.
As you know by now, having seen the stuff above, vmo is geared towards the tuning of VMM.

3.5.1 vmo Housekeeping: vmo flags, and "tuncheck", "tunsave":

First, here is some simplified housekeeping:

```
# vmo -L [tunable}                    # shows the current value.
# vmo -o Tunable=New_Value            # configures the tunable to take on the New_value.
# vmo -p -o Tunable=New_value         # set the New_value now, and also persistently store it in the "/etc/tunables/nextboot".
# vmo -r -o Tunable=New_value         # New_value will be active after the next reboot.
```

So, you might put tunables in files like "/etc/tunables/nextboot", but other files are possible too.
Important is ofcourse, if AIX will execute or check them at boottime. So, a file like "/etc/rc.local" could be
used too to place vmo commands in.

If you want to check a file if at least the ranges of values of your vmo commands are OK, you may want
to check out the "tuncheck" command.

```
tuncheck [ -r | -p ] -f Filename
```

The tuncheck command validates a tunable file. All tunables listed in the file will be checked
for range and dependencies. If a problem is detected, a warning is issued.
There are "two ways" that a check can be done:
- check if the values in a file could be applied immediately, thus in the current context.
- check if the values in the file can be implemented during boot of the system.

-f: used to specify your file with vmo tunables;
-r: check if the file is valid at boot;
-p: checks the file in the current and boot context.

If you want to *save* your current running values of the vmo tunables to a file, then use the tunsave command.

3.5.2 Example: Protecting computational memory from pagestealing.

Your lpar has a certain amount of real memory, which is certainly much much lower than the virtual address space.
So, if you have a number of competing processes for memory pages, the system may start "paging", that is,
swapping pages to and from the page file(s) (or swapfiles).

The paging algolrithm is pretty complex, but anyway, the pagestealing component is very important in this process.

Ofcourse, different purposes of a machine, requires different tunables. For example, for a database machine,
you can optimize VMM in a certain way, which is different from a File Serving machine.

The two most basic page replacement tunable parameters are minperm and maxperm.
These tunable parameters are used to indicate how much memory the AIX kernel should use
to cache non-computational pages.

- The maxperm tunable parameter indicates the maximum amount of memory that should be used to cache non-computational pages.
  But this also influences the point above which the page stealing algorithm steals only file pages.
- The minperm tunable is a measure for the target minimum amount of memory that should be used for non-computational pages.
  But this also specifies the point below which file pages are protected from the re-page algorithm.

So, if you want a FileServer (not stealing pages from filecache), you might do this:

```
# vmo -p -o maxperm%=80                        # actually an AIX 53 default
# vmo -p -o minperm%=60
# vmo -p -o maxclient%=80                      # actually an AIX 53 default
```

But, if you want a Database Server, where you want the pagestealer to leave the computational pages
in place (actually more or less "as much as possible"; it's not exact science), you might do this:

```
# vmo -p -o maxperm%=35
# vmo -p -o maxclient%=35
# vmo -p -o minperm%=15
```

This all also relates to paging. As a sort of "overall" setting, we must turn our attention also to the
lru_file_repage parameter.

The "lru_file_repage" parameter can affect the paging behaviour in considerable way.
By setting this parameter to 0, you force the system to only free file pages when you run out of memory
and to not write working pages out to paging space. This normally will decrease the paging rate,
and %swap-usage.
In other words: AIX, when under memory pressure, will "try harder" to steal memory from the file cache
instead of paging process related pages out to the paging space (lru_file_repage = 0).

So, when dealing with high paging rates, you might want to test the following:

```
# vmo -p -o lru_file_repage=0        # instead of the value 1
```

Chapter 4. Some keypoints on the Object Data Manager - ODM.

The "registry" or "repository", of device, object and application metadata, is called the "ODM" in AIX.
It's quite a unique feature for AIX, and you will not find the "same thing" on most other unixes.
Most prominently, the ODM stores metadata about "devices", and OS related filesets, but for some applications,
(most notably IBM applications like Websphere, VisualAge C etc..), the metadata for that app is stored in the ODM as well.
On the other hand, many other non-IBM software, have "nothing" to do with the ODM. For example, Oracle will use it's
own inventory and depends soley on environment variables, and it's own "registry" files.

ODM has some features of an object database, in the sense that objects are described by classes and attributes.
What is stored in the ODM is: Predefined and Configured device information, Software Vital Product Data, Smit menu's,
information pertaining to the Resource controller, Logical Volume Manager data, NIM, and data for certain (logging) daemons.

4.1 The physical ODM locations and files:

The with ODM corresponding files, can be found in the following directories:

```
/etc/objrepos
/usr/lib/objrepos
/usr/share/lib/objrepos
```

Normally, the admin's $ODMDIR variable points to "/etc/objrepos", so that certain "odm tools"
can operate on the "real" ODM.
If you want to "do something" with the ODM (*which can be dangerous if you do not know exactly
what you are doing*), like extending a schema, you need to do that on a testenvironment first.
On a test lpar, you can copy the ODM files to another location, and set the $ODMDIR to point to the new location.
See section 1.4 for an example.

Some of the most important files are:

Software Vital Product Data:	Devices and LVM:		errdemon:
/etc/objrepos/lpp	PdDv	Predefined Devices	/etc/objrepos/SWservAt
/etc/objrepos/product	PdCn	Predefined Connection	
/etc/objrepos/inventory	PdAt	Predefined Attribute	
/usr/lib/objrepos/lpp	Config_Rules	Configuration Rules	
/usr/lib/objrepos/product	CuDv	Customized Devices	
/usr/lib/objrepos/inventory	CuDep	Customized Dependency	
/usr/share/lib/objrepos/lpp	CuAt	Customized Attribute	
/usr/share/lib/objrepos/product	CuDvDr	Customized Device Driver	
/usr/share/lib/objrepos/inventory	CuVPD	Customized Vital Product Data	

Structure:

/etc/objrepos: Here the customised devices object classes and the SWVPD
(Software Vital Product Database) for the / (root) part of the installable
software product. The directory also contains links to the predefined
devices object classes and the SMIT menu object classes. The ODMDIR
points to here by default.

/usr/lib/objrepos: This contains the predefined devices object classes, the SMIT menu object classes and the four object classes used by the SWVPD for the /usr part of the installable software product. The object classes in this repository can be shared across the network by /usr clients. This can be shared between AIX clients only.

/usr/share/lib/objrepos: This contains the object classes used by the SWVPD for the /usr/share part of the installable software product.

4.2 Devices:

With respect to "devices", the ODM is divided in two sections:

Predefined Devices: PdDv, PdAt, PdCn
Predefined devices - Contains the information of all the devices supported by the operating system. The main files are PdDv (predefined devices), PdAt (predefined attributes), and PdCn (predefined connections).

Customized devices: CuDv, CuAt, CuDvDr
This section stores the information for the devices that are already configured in the system. The main files are CuDv (customized devices), CuAt (customized attributes), and CuDvDr (customized device drivers).

Every device in the ODM has a unique definition that is provided by three attributes:
1. Type
2. Class
3. Subclass

Explanation of how it works:
Most devices are self configuring devices. The command that configures devices is "cfgmgr".
The "cfgmgr" always runs at boottime, but root can also run it at will at any time.
This command verifies the information physically stored in the device (most devices store this information in a ROM), then cfgmgr compares the information against the ODM in the predefined section; if it finds a match, then it creates the entries in the customized section of the ODM. Also, the cfgmgr command loads the driver for the device.
If the device cannot be found in the ODM (predefined) when it is detected, you may need to add the device drivers for that device.
The configuration manager (cfgmgr) runs every time the system is restarted, looking for new devices and verifying the state of the devices previously configured.

If cfgmgr cannot deal with the device (maybe it is not present in the Predefined section), you might create the decription yourself using the "mkdev" command.

Especially the "lsdev" command (more on lsdev later) can show you the status of a device.
- If it is listed as "Available", then it is indeed configured and ready for use.
- If it is listed as "Defined", then the ODM has the description of the device, but it is not configured (yet).

The "devices" ODM files:

- The Configuration Rules (Config_Rules) object class contains the configuration rules used by the Configuration Manager.
- The Customized Attribute (CuAt) object class contains customized device-specific attribute information.
- The Customized Dependency (CuDep) object class describes device instances that depend on other device instances.
- The Customized Device Driver (CuDvDr) object class stores information about critical resources that need concurrence management through the use of the Device Configuration Library subroutines.
- The Customized Devices (CuDv) object class contains entries for all device instances defined in the system.
- The Customized Vital Product Data (CuVPD) object class contains the Vital Product Data (VPD) for customized devices.
- The Predefined Attribute (PdAt) object class contains an entry for each existing attribute for each device represented in the Predefined Devices (PdDv) object class.
- The various bus resources required by an adapter card are represented as attributes in the Predefined Attribute (PdAt) object class.
- The Predefined Devices (PdDv) object class contains entries for all device types currently on the system. It can also contain additional device types if the user has specifically installed certain packages that contain device support for devices that are not on the system.

4.3 Software Vital Product Data

Certain information about software products and their installable options is maintained in the Software Vital Product Data (SWVPD) database. This is considered to be part of ODM.

The SWVPD consists of a set of commands and Object Data Manager (ODM) object classes for the maintenance of software product information. The SWVPD commands are provided for the user to query (lslpp) and verify (lppchk) installed software products. The ODM object classes define the scope and format of the software product information that is maintained.
The installp command uses the ODM to maintain the following information in the SWVPD database:

- Name of the installed software product
- Version of the software product
- Release level of the software product, which indicates changes to the external programming interface of the software product
- Modification level of the software product, which indicates changes that do not affect the external programming interface of the software product
- Fix level of the software product, which indicates small updates that are to be built into a regular modification level at a later time
- Fix identification field
- Names, checksums, and sizes of the files that make up the software product or option
- Installation state of the software product: applying, applied, committing, committed, rejecting, or broken.

4.4 Smit Menu's and dialogs.

The SMIT ODM object repository is under "/usr/lib/objrepos".
Smit uses menu's, dialogs, selectors etc.., which information is stored in:

- sm_menu_opt
- sm_name_hdr
- sm_cmd_hdr
- sm_cmd_opt

In principle (not the subject of this document) you can even extend these class defintions,
to add or distract to or from the functionality of "smit". First, on a test lpar, do something similar as this:

```
$ mkdir ~/objrepos
$ cp /usr/lib/objrepos/sm* ./objrepos
$ ODMDIR=~/objrepos; export ODMDIR            # now $ODMDIR points to your objectrepository
```

Then, invoke smit with the "-o" flag, in order to provide your test ODM location.

For example, to "view" some of those definitions, you can try the following command:

```
$ odmshow sm_menu_opt

class sm_menu_opt {
        char id_seq_num[17];       /* offset: 0xc ( 12) */
        char id[65];               /* offset: 0x1d ( 29) */
        char next_id[65];          /* offset: 0x5e ( 94) */
        vchar text[1025];          /* offset: 0xa0 ( 160) */
        vchar text_msg_file[1025]; /* offset: 0xa4 ( 164) */
        long text_msg_set;         /* offset: 0xa8 ( 168) */
        long text_msg_id;          /* offset: 0xac ( 172) */
        char next_type[2];         /* offset: 0xb0 ( 176) */
etc..
```

4.5 Specific AIX commands to manage or view the objectclasses in ODM.

There are commands, representing an interface to ODM, with which you can add, retrieve, drop and change objects.
The following commands can be used with ODM:

odmget and odmshow,
odmadd,
odmdrop,
odmdelete,
odmcreate,
odmchange

WARNING: It would be a very trivial remark to tell you to be carefull with commands that modify the ODM.

Example 1: odmget, odmshow

The odmget command takes as input a search criteria and a list of object classes,
retrieves the selected objects from the specified object classes, and writes an ASCII odmadd input file to standard output.
But this means also that you can just "view" the objects and attributes, using this command.

```
odmget [-q criteria] objectclass              # where -q criteria is used to filter information.

# odmget -q name=hdisk0 CuAt
# odmget -q 'name=inet0 AND attribute=route' CuAt
# odmget -q 'name=bos.net.tcp.client' lpp
# odmget -q 'lpp_name=bos.net.tcp.client' product
# odmget -q 'lpp_id=74' inventory
# odmget -q name=lv09 CuAt                    # the "lv09" object is a Logical Volume (in chapter 4 more on LV's)
```

```
CuAt:
        name = "lv09"
        attribute = "lvserial_id"
        value = "00d7beec00004c00000001249820e17f.19"
        type = "R"
        generic = "D"
        rep = "n"
        nls_index = 547
etc.. (rows omitted)
```

The odmshow command takes as input an object class name (ObjectClass) and displays
the class description on the screen. The class description is in the format taken as input to the odmcreate command.

```
odmshow objectclass

$ odmshow sm_menu_opt
```

Example 2: odemdelete, odmadd

-- odmadd:

The following command will create entries in the ODM of a certain ibm storage system.
After the predefined sections are available, and the storage system is accessible,
the cfgmgr might find this device and create customized entries.

```
# odmadd array.iscsi.ibm-dac-V4.add
```

-- odmdelete:

Normally, when you delete a device using smitty or rmdev, that would do the job.
Also, uninstalling an application, using the right uninstaller, would do the job too.

But, sometimes, an uninstall of some app fails, thereby leaving entries in the ODM.
In some of those cases, you might need to clean the ODM from those unwanted entries.

Also, some regular commands will issue a number of "odmdelete" statements, like using
"exportvg" if you want to export a Volume Group from your system.

As an example using "odmdelete", consider the following script.

Suppose you know there are incorrect ODM entries of the rootvg.
Then you can try to use the "rvgrecover" shell script.

The script "rvgrecover" issues a series of odmdelete statements, just like exportvg does.
At the end of the script, an importvg is done.
The importvg command, reads the VGDA and LVCB from the boot disk, resulting in new ODM entries.
This script, is only provided for illustration purposes of using "odmdelete".
The rvgrecover script has the following contents:

```
PV=/dev/ipldevice  # PV=hdisk0
VG=rootvg
    cp /etc/objrepos/CuAt /etc/objrepos/CuAt.$$
    cp /etc/objrepos/CuDep /etc/objrepos/CuDep.$$
    cp /etc/objrepos/CuDv /etc/objrepos/CuDv.$$
    cp /etc/objrepos/CuDvDr /etc/objrepos/CuDvDr.$$
    lqueryvg -Lp $PV | awk '{ print $2 }' | while read LVname; do
        odmdelete -q "name = $LVname" -o CuAt
        odmdelete -q "name = $LVname" -o CuDv
        odmdelete -q "value3 = $LVname" -o CuDvDr
    done
    odmdelete -q "name = $VG" -o CuAt
    odmdelete -q "parent = $VG" -o CuDv
    odmdelete -q "name = $VG" -o CuDv
    odmdelete -q "name = $VG" -o CuDep
    odmdelete -q "dependency = $VG" -o CuDep
    odmdelete -q "value1 = 10" -o CuDvDr
    odmdelete -q "value3 = $VG" -o CuDvDr
    importvg -y $VG $PV      # read disks metadata in order to get the ODM entries
    varyonvg $VG
```

No need to understand this script completely. The point is that we see that "odmdelete" operates on CuAT, CuDv, CuDep etc.. In order to remove all entries from ODM. That's all.

4.6 Regular AIX commands to view the ODM objects, attributes and values:

Device configuration:	lsdev, lsattr, lscfg
Software:	lslpp, lppchk

Although it depends "on where you are", that is, a normal lpar, or a virtual IO Server (VIOS), these commands can display different context.

In all cases, the commands will show you ODM objects, attributes and values.

Examples:

lsdev:

The lsdev command displays information about devices in the Device Configuration database.
You can display the default output one of the following ways:

- From the Customized Devices object class using the -C flag
- From the Predefined Devices object class using the -P flag

```
# lsdev -Cc tape
rmt0  Available  10-60-00-5,0  SCSI 8mm Tape Drive
```

```
# lsdev -Cc disk
hdisk0 Available 20-60-00-8,0    16 Bit LVD SCSI Disk Drive
hdisk1 Available 20-60-00-9,0    16 Bit LVD SCSI Disk Drive
hdisk2 Available 20-60-00-10,0   16 Bit LVD SCSI Disk Drive
hdisk3 Available 20-60-00-11,0   16 Bit LVD SCSI Disk Drive
hdisk4 Available 20-60-00-13,0   16 Bit LVD SCSI Disk Drive
```

```
# lsdev -Cc adapter
ent0      Available 04-08 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
ent1      Available 04-09 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
ent2      Available 07-08 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
ent3      Available 07-09 2-Port 10/100/1000 Base-TX PCI-X Adapter (14108902)
fcs0      Available 05-08 FC Adapter
fcs1      Available 09-08 FC Adapter
ide0      Available 03-08 ATA/IDE Controller Device
lai0      Available 0B-00 GXT135P Graphics Adapter
ohcd0     Available 02-08 USB Host Controller (33103500)
ohcd1     Available 02-09 USB Host Controller (33103500)
sa0       Available       LPAR Virtual Serial Adapter
sa1       Available       LPAR Virtual Serial Adapter
sisscsia0 Available 08-08 PCI-X Dual Channel Ultra320 SCSI Adapter
sisscsia1 Available 06-08 PCI-X Dual Channel Ultra320 SCSI Adapter
sisscsia2 Available 0K-08 PCI-XDDR Dual Channel Ultra320 SCSI Adapter
```

The following commands, lsattr and lscfg, usually uses the "-l" flag to name a particular device from which you want information.

lsattr:

Displays *attribute characteristics* and possible *values of attributes* for devices in the system.

You must specify one of the following flags with the lsattr command:

-D Displays default values.
-E Displays effective values (valid only for customized devices specified with the -l flag).
-F Format Specifies the user-defined format.
-R Displays the range of legal values.

Most of the time, you would use "-E" in order to show the running values, and "-l" to give as a parameter the device logical name.

This command gets the current attributes (-E flag) for a tape drive:

```
# lsattr -E -l rmt0
mode           yes     Use DEVICE BUFFERS during writes    True
block_size     1024    Block size (0=variable length)      True
extfm          no      Use EXTENDED file marks             True
ret            no      RETENSION on tape change or reset   True
..
```

To list the default values for a tape device (-D flag of "Default"), use

```
# lsattr -l -D rmt0
```

This command gets the attributes for a network adapter:

```
# lsattr -E -l ent1
busmem      0x3cfec00     Bus memory address     False
businttr    7             Bus interrupt level    False
..
```

To list only a certain attribute (-a flag), use the command as in the following example:

```
# lsattr -l -E scsi0 -a bus_intr_lvl
bus_intr_lvl 14 Bus interrupt level False

# lsattr -El tty0 -a speed
speed 9600 BAUD rate true
```

lscfg:

Displays configuration, diagnostic, and vital product data (VPD) information about a device, or the system.

lscfg [-v] [-p] [-s] [-l Name]

This command gets the Vital Product Data for the tape drive rmt0:

```
# lscfg -vl rmt0
Manufacturer...............EXABYTE
Machine Type and Model.....IBM-20GB
Device Specific(Z1)........38zA
Serial Number..............60089837
..
```

```
-l Name Displays device information for the named device.
-p Displays the platform-specific device information. This flag only applies to
   AIX 4.2.1 or later.
-v Displays the VPD found in the Customized VPD object class. Also, on AIX 4.2.1
   or later, displays platform specific VPD when used with the -p flag.
-s Displays the device description on a separate line from the Name and
   location.
```

```
# lscfg -vp | grep -p 'Platform Firmware:'

# lscfg -vp | grep -p Platform

Platform Firmware:
ROM Level.(alterable).......3R040602
Version....................RS6K
System Info Specific.(YL)...U1.18-P1-H2/Y2
Physical Location: U1.18-P1-H2/Y2
The ROM Level denotes the firmware/microcode level
Platform Firmware:
ROM Level ............. RH020930
Version ................RS6K
..
```

The following command shows details about the Fiber Channel cards:

```
# lscfg -vl fcs*          (fcs0 for example, is the parent of fsci0)
```

Chapter 5. Some keypoints on the system p environment.

Before we can go into a subject like "virtualization", "PowerVM", "LPARS" etc..
we need to have a "little overview" on the system p environments itself.

What machines are we talking about when we consider AIX?
And, can that hardware run other OS as well?

There are many platforms using Power architecture, for example "iSeries" or "system i" (originally used for i5/OS),
and "pSeries" or " system p" (originally for AIX, and Linux).
Since Power5, the Hypervisor in "I" and "p" are very similar, so running AIX and i5/OS (AS400 descendant)
can be done in different LPARS.
System p as of Power5, can implement a shared pool of cpu's, and make "micropartitioning" possible.
Actually, that feature was already available on "system i" on Power4.
System p on Power4, used dedicated resources per lpar (like dedicated cpu).

From now on, if we talk about Power, it should be understood that we are talking about Power on "p".

Here are some further facts:

- The machine architecture + processor architecture + Virtualization options, are "somewhat" summed up
  (or expressed) by the "POWER platform", like POWER4, POWER5, and the latest POWER6 architecture.
  Ofcourse, the used Processors *themselves*, determine the Power version *in the first place*, but
  a certain Power version, implies what your virtualization options are as well.

- A physical Machine, usually get's partioned into "Virtual Machines" or "LPARS's " (Logical Partitions),
  where each LPAR is a full OS installation, including such properties as having a Hostname, IP adress,
  and in case of AIX, a ROOTVG (with "/", "/usr", "/etc" and other typical OS filesystems), and possibly other VG's.
  As from AIX v6 (and not related to Power), it is possible to create confined "containers", called WPAR's,
  running <u>inside</u> an AIX lpar. These WPARS, are shielded subsets, for running specified applications.

- A major feature of the POWER5 machines is a new Hypervisor that convergenced with iSeries systems
  As a result, on Power5/6 on p, you can run:
  -AIX (was originally the target OS for a system p lpar)
  -i5/OS (or ibm I, the AS/400 descendant)
  -Linux for Power (Suse, RedHat)

System p (pSeries) machine models (up to 2009):

The physical p machines, have had various names like "eServer pSeries", but their latest "mark" is "system p".
Some of the main models of the recent "past", and today's models, are:

The Large range of RS/6000 models. Power2/3
typical version of AIX used at later models: 4.3

Some Former Models (Power4/4+)
pSeries 610
pSeries 615
pSeries 620
pSeries 640
pSeries 655
pSeries 670
pSeries 680
pSeries 690
Typical AIX: v5.1, v5.2

Recent models (Power5/5+):	
p 510	entry model
p 520	low-end midrange
p 550	low-end midrange
p 560	middle midrange
p 570	middle midrange
p 575	middle midrange
p 590	high-end
p 595	high-end
Typical AIX: v5.3, v6.1	

Latest "converged" system p 2009 model	
system (Power6/6+)	MTM machine type model
Power 520	Unified 8203-E4Aa
Power 550	Unified 8204-E8Aa
Power 570	Unified 9117-MMAa
Power 595	Unified 9119-FHAa
Typical AIX: v5.3, v6.1	

So, what you can encounter as an OS on system p is most likely to be:
AIX, or Linux, and (sometimes) i5/OS (or "IBM I")
But the most typical OS found on system p, still is AIX (so it seems to me).

Characteristics of POWER Virtualization:

POWER4 Partitioning is mainly characterized by the fact that a LPAR used dedicated processor(s), fixed memory,
and dedicated physical IO adapters. At any time, resources (like cpu) could be added or removed.
DLPAR, or Dynamic LPAR, made it possible to add or remove resources without restarts.
Although the "theory" was pretty clean, I have to say that it was not so easy to implement it.

POWER5 Partitioning is mainly charcaterized by the fact that "micropartitioning" is possible:
LPARS are allocated processor resources from a shared processor pool.

By enabling special LPAR(s), called Virtual IO Server (VIOS), IO resources can be "virtualized", resulting in
"virtual SCSI" and "virtual Ethernet".
It is still possible however, to allocate dedicated CPU('s), (and IO adapters) to an LPAR, which then do not participate
in the shared processor pool.
And, for an lpar to be able to use micropartitioning, you need at least AIXv5.3. Lower versions uses the
the dedicated resource model.
Furthermore, virtualized devices can only be assigned to AIX v5.3 or later.

POWER6 builds further on POWER5, with a lot of additional features like "PowerVM Active Memory Sharing",
and in combination with AIX v6, WPAR or Workload Partitioning (resulting in confined application "containers"),
running as "subsets" within an LPAR. Also, "live mobility" can be done, which is the ability to migrate
an active or inactive AIX or Linux logical partition from one system to another.

	CPU per lpar	Memory per lpar	IO per lpar
POWER4 on p	dedicated CPU (granularity 1)	dedicated memory (granularity 256M)	dedicated adapter
POWER5 on p	- Shared pool of cpu's virt. cpu's: min/max/desired - 0.01% ent. cpu: min/max/desired - 0.1% - dedicated CPU still possible	dedicated memory, but with: min/max/desired	dedicated adapter still possible Virtualized SCSI with VIOS Virtualized Ethernet with VIOS
POWER6 PowerVM on p	- Shared pool of cpu's virtual cpu's: min/max/desired entitled cpu: min/max/desired - dedicated CPU still possible	dedicated memory, but with: min/max/desired Active Memory Sharing	dedicated adapter still possible Virtualized SCSI with VIOS Virtualized Ethernet with VIOS

Frame:

The 'whole" of a "system p" machine, is most often referred to as "frame".
A frame can consists of multiple components, like powerunits, IO drawers, and most notably
the "CEC", which is the "Central Electronics Complex".
The CEC will contain a number of "processor books".

Some typical numbers on the number of LPARS:

- A p520 is often not partitioned (just running the "full system partition").
  Or it runs a couple of LPARS (not withstanding the fact, that some people push them to the very limits).
- A p550 probably has a couple of LPARS (say 4, 6 or 8).
- A p570 usually has about 12, 16 or 24 LPARS (or so) configured.
- A p595 could have so many resources, that a high number of LPARS might live there.

But, in all cases, numbers are just some typical values, and there could be LPARS with very heavy resource requirements
(big database Server, high-end app server etc..), that might easily lower the total number of LPARS.

The Hardware Management Console:

As of 550, all p machines have a HMC connected (although all p machines types could have a HMC connection).
It's a graphical console (based on Linux).
The HMC is mainly used in configuring and maintenance of LPARS.
It actually has "Wizards" in helping you setting up LPARS, displaying all necessary screens.

Along with setting up LPARS, after they are up and running, you can use (in a fully graphical way) the HMC to:
- shutdown and restart an LPAR
- Reconfigure resources like cpu, memory, virtualized adapters, per LPAR
- Install and configure VIOS
- Install the OS in an LPAR

The HMC is also used to view and manage the p machine itself.
One aspect of that, is the "Service Focal Point" for receiving, logging, and tracking system errors,
and even possibly (or likely to be) configured for reporting automatically critical events to IBM services.

The are a few alternatives (or add on's) like IVM, WebSM.

The Virtual IO Server (VIOS):

Virtual I/O Server can provide "virtual SCSI" and "virtual networking" to LPAR clients.
The VIO Server (or VIOS) can thus save on using private adapters at LPARS.
Only clients "of sufficient level" can use the through VIO exposed resources.
For AIX it means *as of* version 5.3 can use VIOS exposed adapters. So, AIX 5.2 LPARS must use dedicated adapters.

The Hypervisor:

Central to Virtualization is the Hypervisor.
the POWER Hypervisor is for example responsible for dispatching all the workloads of all active LPARS across the physical processors.
It also provides for inter-partition communication that enables the Virtual I/O Server's virtual SCSI and virtual Ethernet function.

Embedded in firmware, the Hypervisor is always present and can be viewed as a layer between the guest Operating Systems running in LPARS, and the hardware.

There is quite a difference between former AIX versions, and version v.5.3.
As you can see from below output of "topas -L", the OS can make "hcalls" which are calls to the Hypervisor, not unlike systemcalls in an OS. It's a fully Hypervisor "aware" operating system.

```
# topas -L

   Interval:   2            Logical Partition: lp05       Wed Dec 23 19:24:15
        Poolsize: 3.0                 Shared SMT ON            Online Memory:   8192.0
        Entitlement: 2.5               Mode: Capped            Online Logical CPUs:  4
                                                               Online Virtual CPUs:  2
        %user  %sys  %wait  %idle physc  %entc %lbusy   app   vcsw  phint %hypv hcalls
         47.5  32.5    7.0   13.0   2.0   80.0  100.0   1.0    240    150   5.0   1500
        ==============================================================================
        logcpu minpf majpf  intr   csw  icsw runq lpa  scalls usr sys  wt idl  pc lcsw
        cpu0    1135   145   134    78    60    2  95   12345  10  65  15  10 0.6  120
        cpu1     998   120   104    92    45    1  89    4561   8  67  25   0 0.4  120
        cpu2    2246   219   167   128    72    3  92   76300  20  50  20  10 0.5  120
        cpu3    2167   198   127    62    43    2  94    1238  18  45  15  22 0.5  120
```

Toolbox Linux:

If you have installed AIX in some LPAR, and you want to use a linux toolset, then that's possible if you install the "AIX Toolbox for Linux Applications".
The AIX Toolbox for Linux Applications contains free open source software built for AIX 5L, packaged with RPM. It includes a variety of utilities & libraries often found on Linux distros.

Most of the software will be installed in "/opt/freeware".

Chapter 6. Some keypoints on Virtualization.

Very Thick Fat redbooks have been written on this subject, so what can a single simple chapter of some simple guy tell you?
Well.. Not much indeed. But some important keypoints should be possible.

6.1 Quick Overview:

What we are trying to achieve, is "somewhat" (I am not such a good artist) illustrated by the figure below.
The Redbooks have much clearer pictures ofcourse, but I don't want to copy those ;-)

What I hope to make clear here, is that the AIX v5.3 and v6.1 LPARS are "micropartitioned"
meaning that their CPU resources are coming from the "shared pool" of CPU, and can be fractions of cpu units.
Here, the AIX 5.3 LPAR, has an "entitled" cpu capacity of 0.3.
The AIX 6.1 LPAR, has an "entitled" capacity of 1.5.
Contrary, the AIX 5.2 LPAR cannot be micropartitioned, and must used "dedicated" cpu's.
Note: If you want, the 5.3 an 6.1 LPARS could have used dedicated CPU as well, but that is not common practice.

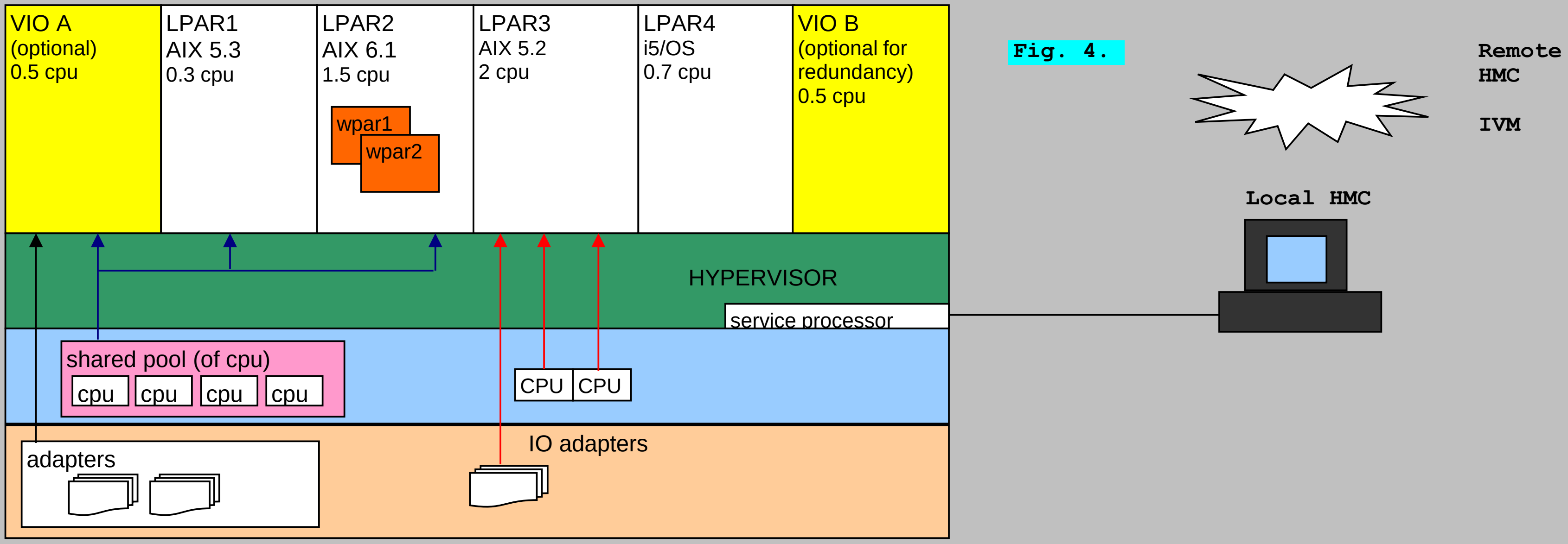


Fig. 4.

The VIO's are optional. These "Virtual IO Servers", are specialized LPARS, for managing and exposing
"virtual storage" and "virtual ethernet". Again, only AIX client LPARS as of v5.3 (and Linux, and i5/OS)
can be "clients" of those resources.
Hopefully you can see from the figure, that the AIX 5.2 client LPAR, uses "dedicated adapters",
while the AIX 5.3 and 6.1 LPARS uses the "virtualized disk and ethernet resources", as it is
provided bij the VIO Server.
Again, I hope it get's clear from the figure, that LPAR3 (the AIX5.2 LPAR) uses dedicated adapters,
but that a whole bunch of other IO adapters (mass storage devices, Ethernet) are "owned" by the VIOS.
The VIOS has very sophisticated ways to map those resources to client LPARS, which then can use them.

VIO thus may be used to reduce the need for private, dedicated, adapters.

If you want to use those virtualized resources as adapters, storage, and networking, then you must use
at least one VIO Server. But to avoid a single point of failure, with production systems, you
will add one VIOS for redundancy. That might even go further one step, by creating 2 VIOS for networking
purposes, and 2 VIOS for providing virtualized storage.
So, in such a case, on a larger production machine, you might see a SANVIOA, SANVIOB, NETVIOA, NETVIOB.
So, here VIOS'ses are added for redundancy, and functionality is split up in virtual network and virtual storage.

*Note: suppose the machine in figure 4, was a p570, probably something like 16 (or so) LPAR's
would have been configured. Also, in reality, a "wonderfull" mix of different AIX versions, and i5/OS, and Linux LPARS
is not so very likely. Most of the time, an Organization obviously wants standardization of versions and OS.*

The HMC is the graphical console, with which you manage your "system p" machine, and the LPARS that live on it.
As you may have read in Chapter 5, the HMC is used mainly to manage LPARS, that is,
creating LPARS, configure them to use certain amounts of resources, manage them (like shutdown/startup),
install Operating Systems in the LPARS, and re-configure LPARS with respect to resources as needed.

Usually, a "local" HMC is present, physically located near the managed system p machine(s). But "remote" HMC consoles are
possible too.
The local HMC needs a network connection to the "service processor" of the managed 'system p' machine.
Note: to manage Power6 machines, you need HMC v 7.
It's very likely that a HMC is used in your work. But (especially meant for smaller systems), an alternative
is the Integrated Virtualization Manager (IVM), which is webbased. IVM is packaged in the Virtual I/O Server (VIOS) software.

Usually, an IBM engineer have set things up, like testing a freshly installed system p machine, and connecting it to the HMC.
I said "usually", because that is a likely scenario with the larger (and expensive) machines like the p590.
But, with the smaller machines, maybe you as as sysadmin have to set things up. And, maybe you do that as well
with the large systems.

Anyway: creating and configuring the LPARS really is your business. Also, installing software on those LPARS,
is one of your duties too.
The default userid and password of a HMC is: userid=hscroot, password=abc123
Ofcourse, at any live HMC, that password has been changed soon after the installation.

6.2 Planning, and assigning resources (cpu, memory) to LPARS:

Remember, older AIX versions only use dedicated CPU (at granularity of 1).
As of AIX v5.3, micropartitioning can be implemented.

We can create LPAR's using commands (or scripts) from the HMC.
However, since the HMC provide graphical "Wizards" to create LPARS, many sysadmins
will just use the relatively easy graphical console.

How you will do the partitioning, using commands/scripts, or in a graphical way, probably
also depends on the size of your "shop". If you have a couple of 570's, or a 590,
you may choose the easy graphical setup. If you work in a sort of "assembly line", that have
to produce hundreds of LPAR's, you probably will create scripts.

However you will do partitioning, you need to do a certain amount of PLANNING beforehand.
You always should know the "needed" capacities beforehand, that you are about to assign to the LPARS.

Although certain features like "capped/uncapped" and " min / max / desired" of a resource (like cpu, memory)
can be associated with an lpar (and can eliviate potential problems), you still need to know what the LPARS are for.
Suppose you have created 16 LPARS on a 570, all configured with a (pretty low) mean value of 0.3 cpu, 2GB memory,
and suddenly it turns out that (say) 2 LPARS actually are not just "middle of the road" Servers, but one should function
as a HUGE database Server, and the other should be a HUGE application server, then.. You may have a problem.

So, planning and knowing what the LPARS are for, is critical.

Using VIOS is not always "all sunshine" either. If some LPARS have high Backup & Recovery requirements,
using virtualized resources as network and storage, the capacity of the shared VIOS resources may
not be large enough to serve the bandwith of all backing up all LPARS.
Carefull considerations and planning is essential here, for production systems.

Setting values at Micropartitioning:

When setting up an LPAR using micropartitoning, either by commands/scripts or graphical, the following
attributes will be encountered (either shown in a graphical screen, or as a parameter on the commandline):

PU: Processing Units:

Processing capacity can be configured in fractions of 1/100 of a processor. The
minimum amount of processing capacity that has to be assigned to a partition is
1/10 of a processor.
On the HMC, processing capacity is specified in terms of "processing units".
So, for example, you could allocate a LPAR a PU value of 0.8.
In this case, the LPAR is "entitled" a CPU capacity of "0.8 CPU".

When setting up an LPAR, you can specify a Minimum (min), a Desired (Des), and a Maximum (max)
value of PU. Then, when the Hypervisor is distributing the total workload, you LPAR will not
dive under the Min value, and it will not exceed the Max value. Normally, it will operate in
the "Desired" value.

If you were actually setting up an LPAR, these settings would be stored in a "partition profile",
(as well as what you would select as settings for virtual cpu's, and memory).

Virtual processors:

It's a representation of a entity that behaves like a cpu. Assigning more virtual processors
to a LPAR, enhances "concurrency".

Here too, when setting up an LPAR, you can specify a Minimum (min), a Desired (Des), and a Maximum (max)
value of the number of virtual cpu's. Then, when the Hypervisor is distributing the total workload, you LPAR will not
dive under the Min value, and it will not exceed the Max value. Normally, it will operate in
the "Desired" value.

Min, Max, Desired Memory:

The same principle applies with memory too. You can specify a Minimum, a Maximum, and Desired amount of memory.

Capped, Uncapped:

Capped: The assigned PU to a partition will never exceed the max PU, even if resources are
available in the shared pool.

Uncapped: The assigned PU to a partition, may exceed the max PU, if resources are available.
Since you may have multiple LPARS to be "uncapped", specifying a "weight" for this LPAR
must be done too. So, if multiple LPARS are uncapped, then the system will know how to distribute
free resources to them.
The weight can also be used to favor one LPAR, over the others.

Partition profiles:

When working with the HMC, the values for the above atributes (like min, des, max PU) will be saved to a profile.
You could create several profiles, and when you would start an LPAR, you can then choose which profile to use.
The first profile you create (when setting up an LPAR), is called the "default profile".

Document and calculate it:

A spreadsheet (or other type of registering information) can be of help.
For a "system p" machine, you could create a document like the following example:

	Virt. Cpu			PU			Memory			Capped	Weight
LPAR:	min	des	max	min	des	max	min	des	max		
VIOA											
VIOB											
N05L01											
N05L02											
N05L03											
etc..			totals:	sum	sum	sum	sum	sum	sum		

6.3 A few remarks on creating a LPAR using the commandline on the HMC:

One important item should be clear. You <u>first</u> create a LPAR (of a certain "type", like VIO or AIX etc..).
Then, <u>when the LPAR exists</u>, then you install the corresponding Operating Software on it (like AIX).

Let's see how we can create a LPAR using commands on the HMC.

Many commands and options are available using the HMC.
You can use commands for a variety of things like:
- for documentation or query purposes
- influence the state of an lpar (stop, start, use another profile etc..)
- creating objects, like a LPAR

>> Query for information:
For example, you can use the "lssyscfg" command to obtain information from all lpars on a managed system.
This command has nummerous flags. Use the "-m" flag to specify a "system p" frame.

```
 hscroot@hmc-op:~>  lssyscfg -m pSRV570 -r lpar -F name,lpar_id,state
 N05LP06,6,Running
 N05LP05,5,Running
 N05LP04,4,Running
 N05LP03,3,Not Activated
 N05LP02,2,Running
 VIO-ServerA,1,Running
```

>> Changing the state of a LPAR:
For example, to shutdown a certain LPAR on a managed system, you can use:

```
hscroot@hmc-op:~> chsysstate -r lpar -m pSRV570 -o shutdown -n N05LP04
```

>> Create a new LPAR:
For example, to create a new LPAR, use a command similar to:

```
hscroot@hmc-op:~>  mksyscfg -r lpar -m pSRV570 -i name=n05LP07, profile_name=normal, lpar_env=aixlinux,
 min_mem=512, desired_mem=2048, max_mem=4096,   proc_mode=shared, min_proc_units=0.2, desired_proc_units=0.5,
 shared_proc_pool_util_auth=1, max_proc_units=2.0, min_procs=1, desired_procs=2, max_procs=2, sharing_mode=uncap,
 uncap_weight=128, boot_mode=norm, conn_monitoring=1, shared_proc_pool_util_auth=1
```

It's only an example. But these are pretty complex commands indeed.
Do you notice the min/max/des values for the memory and PU's?

Mind you: this is ONLY setting up the LPAR (of a certain type, like an AIX or Linux capable LPAR).
After the LPAR exists, you still need to install the Operating System.

6.4 A few remarks on Creating a LPAR the graphical way:

You can create different types of LPARS:

1. VIO Server
2. I5/OS LPAR (an LPAR supporting the i5/OS, or "ibm I" Operating System - the AS/400 descendant)
3. An AIX-, or Linux LPAR

That you want to create 2) or 3) is rather obvious. These are the functional Operating Systems, and only you know which Operating Systems you need to install on those LPARS.
Note: for i5/OS (ibm I) lpars, some licensing constraints exists.

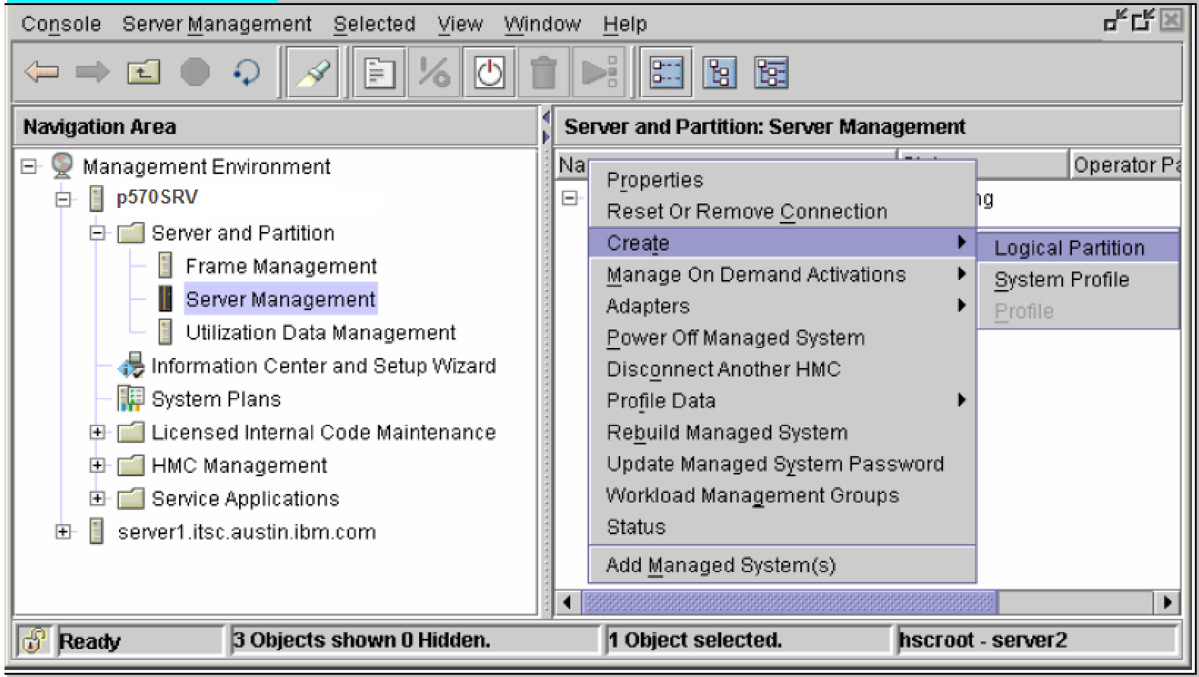
Only if you want to implement "virtual scsi", and/or implement "shared ethernet",
then you need at least one VIOS (and possibly two for avoiding a single point of failure).

This is how you normally proceed:

- Create the LPAR of the type you want.
- After the LPAR is created, install the correct OS (Like AIX, Linux, or VIO Server software).

So, using the graphical way, right-click "Server Management", then choose "Create", and
next choose "Logical Partition" (see Fig. 5).

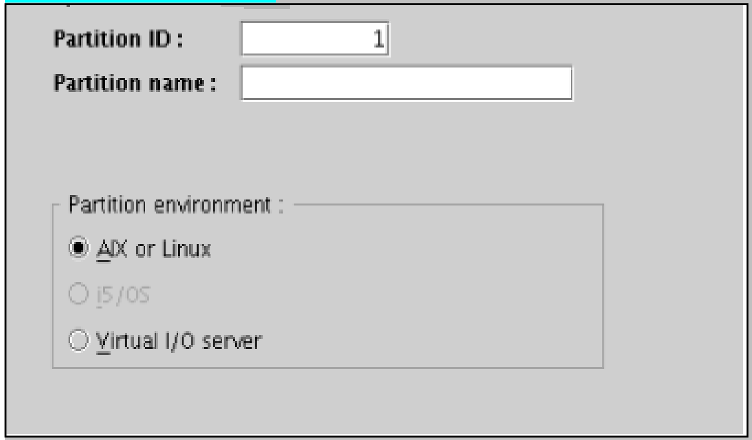
Fig. 5.



In the screen that follows (shown in figure 6), you must specify

- a "Partition ID" (the next number, or if this is the first LPAR, it's "1"),
- the Name of the LPAR (Like for example, VIOSA, or N05L06)
- and, make the choice whether this LPAR is going to be a: AIX or Linux lpar, or a i5/OS lpar, or a Virtual IO Server.

Fig. 6.



Partition ID : 1

Partition name :

Partition environment :

[✓ AIX or Linux]

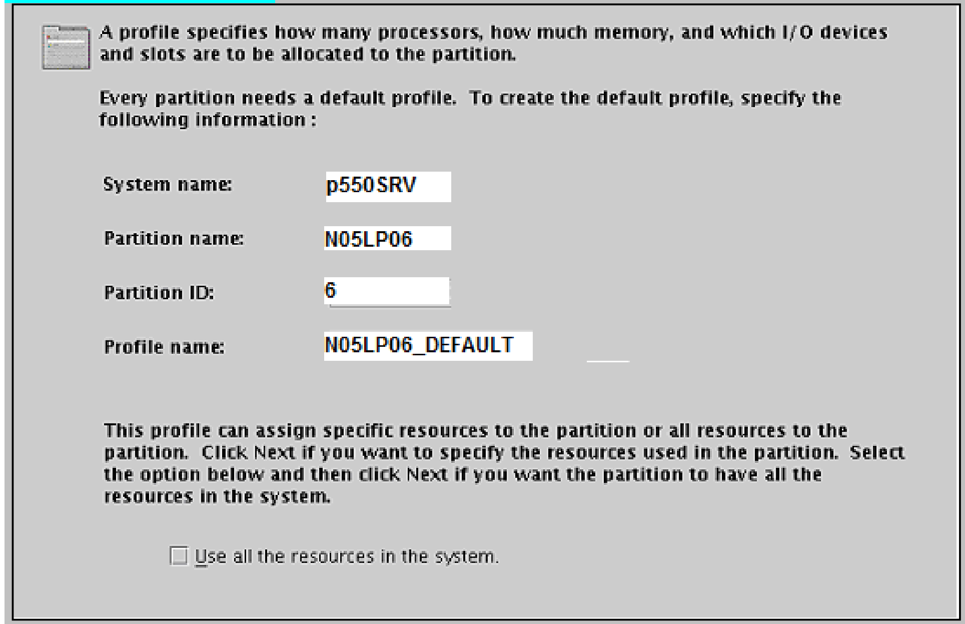
i5/OS

Virtual I/O server

If this would be your "starting point" in setting up a system p, and you would like to use "virtual scsi",

you probably would create the VIO LPAR first (it's not mandatory, you can always do it later).
Suppose that we just want to create a "normal" AIX LPAR. We can always create a VIOS at a later point.
After making our choices in figure 6 (like for example Partition ID=6, Partition Name=N05LP06, and it's an AIX LPAR),
the following screen appears:

Fig. 7.


A profile specifies how many processors, how much memory, and which I/O devices and slots are to be allocated to the partition.

Every partition needs a default profile. To create the default profile, specify the following information :

System name:	p550SRV
Partition name:	N05LP06
Partition ID:	6
Profile name:	N05LP06_DEFAULT

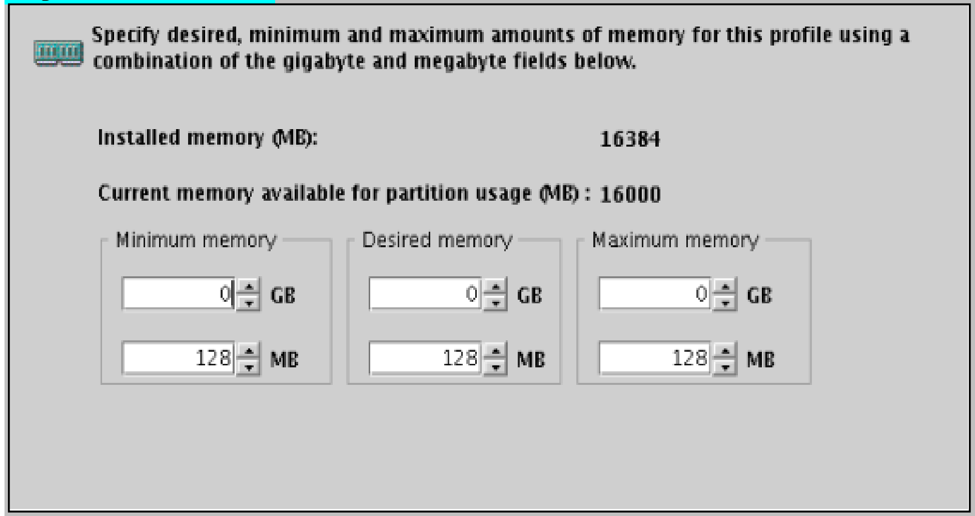
This profile can assign specific resources to the partition or all resources to the partition. Click Next if you want to specify the resources used in the partition. Select the option below and then click Next if you want the partition to have all the resources in the system.

☐ Use all the resources in the system.

All the settings you specify in the following screens, like min/desired/max PU (see section 6.2)
and other parameters, will be stored in a "partition profile".
That's why you need to give a "profile name" here, because you are able to create other profiles as well.

What then will follow, are setup screens where you can specify min/desired/max Memory, and the same for PU,
and the same thing for assigning virtual cpu's. This is what we discussed in section 6.2.

Fig. 8.


Specify desired, minimum and maximum amounts of memory for this profile using a combination of the gigabyte and megabyte fields below.

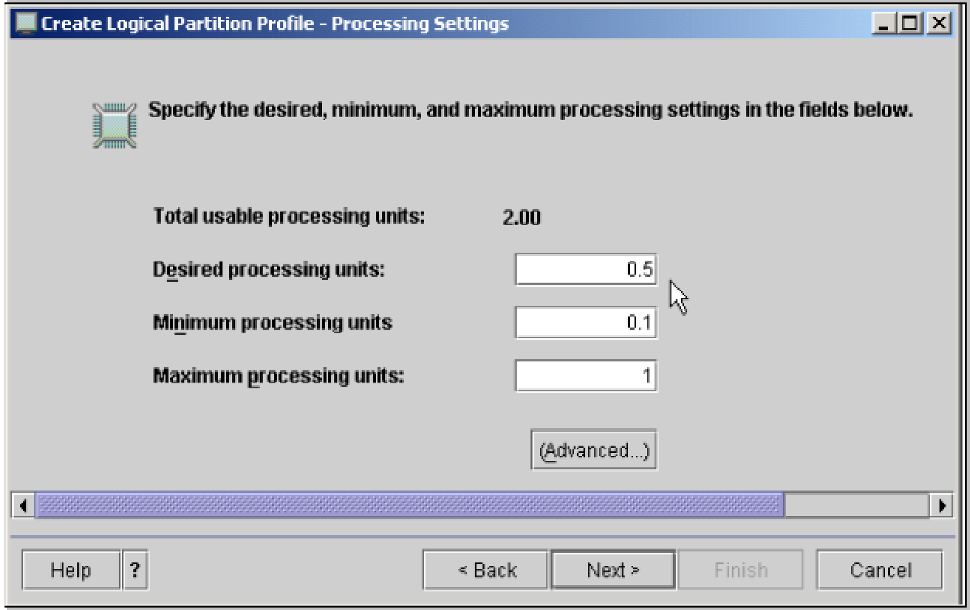
Installed memory (MB): 16384

Current memory available for partition usage (MB) : 16000

	Minimum memory	Desired memory	Maximum memory
GB	0	0	0
MB	128	128	128

In fig. 8, you can say what you want for "min/desired/max" Memory values for this particular LPAR.
It works quite the same way for PU and virtual cpu's.

Fig. 9.


Create Logical Partition Profile - Processing Settings

Specify the desired, minimum, and maximum processing settings in the fields below.

Total usable processing units: 2.00

Desired processing units: 0.5

Minimum processing units 0.1

Maximum processing units: 1

A similar screem will follow, for setting the min/desired/max of virtual cpu's.

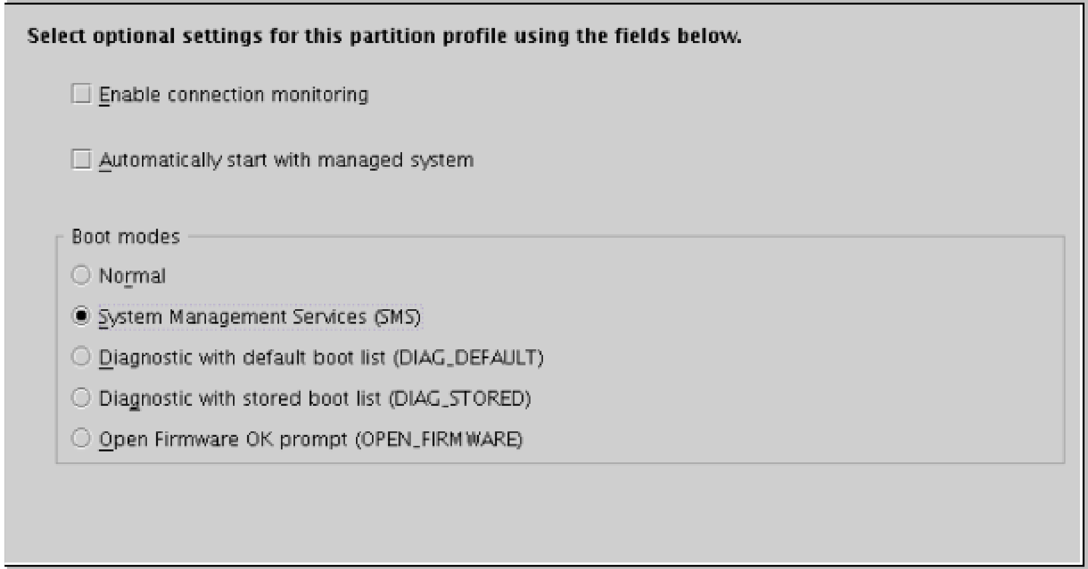
You understand that we are trying to setup micropartitioning (as of AIX 5.3), and thus
we do not need "dedicated cpu('s)", but instead, we can make use of the shared pool of cpu's,

which enables us to specify "min/desired/max PU" and "min/desired/max virtual processors".

Anyway, this section and the former one, gave a little insight on how to create LPARS, using the commandline (of HMC), and using a graphical setup Wizard (of HMC).
At some point, after the LPAR(s) are created, we still need to install the OS'ses on them.

So, we skip a number of setup screens, and at some point you are able to set the "bootmode":

Fig. 10.


Select optional settings for this partition profile using the fields below.

☐ Enable connection monitoring

☐ Automatically start with managed system

Boot modes

○ Normal
● System Management Services (SMS)
○ Diagnostic with default boot list (DIAG_DEFAULT)
○ Diagnostic with stored boot list (DIAG_STORED)
○ Open Firmware OK prompt (OPEN_FIRMWARE)

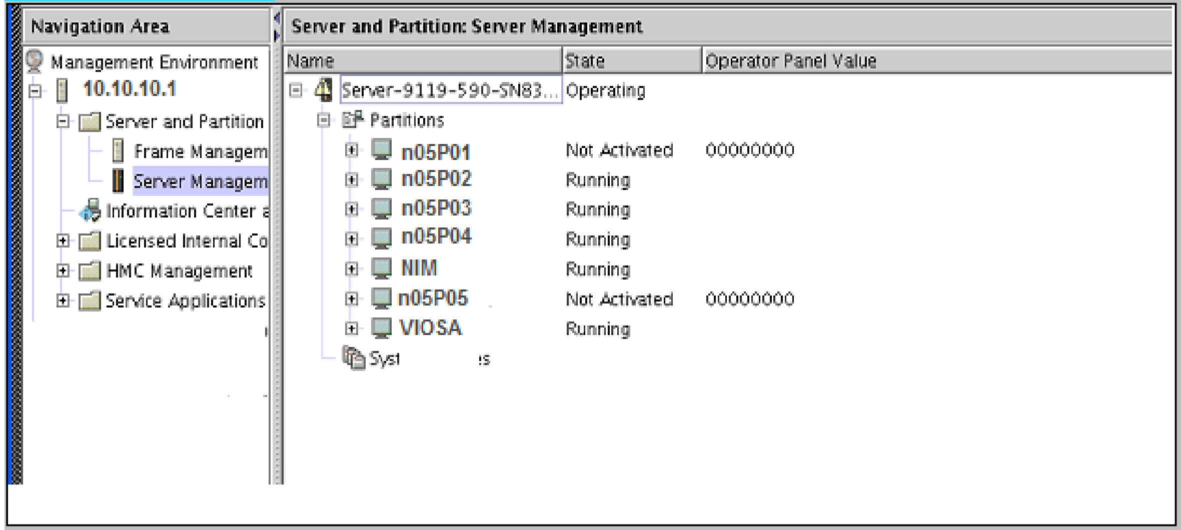
Ofcourse, after the OS is loaded and operational in the LPAR, the bootmode should be set to "normal", which means a normal boot.

But, at this point, while we are configuring the LPAR, we should put it into "SMS bootmode", which enables use to install the Operating System at the next start of the LPAR.
This could be a network installation of the OS using "NIM", or using a local CD / DVD.

Independent of the fact that you were setting up a regular LPAR (like AIX, or Linux), or you were setting up a VIOS, the "red line" in the sketched setup presented above, is quite same.

After you have setup a VIOS, and some client LPARS, and all Operating systems would also have been installed, you might see something similar as the following HMC screen:

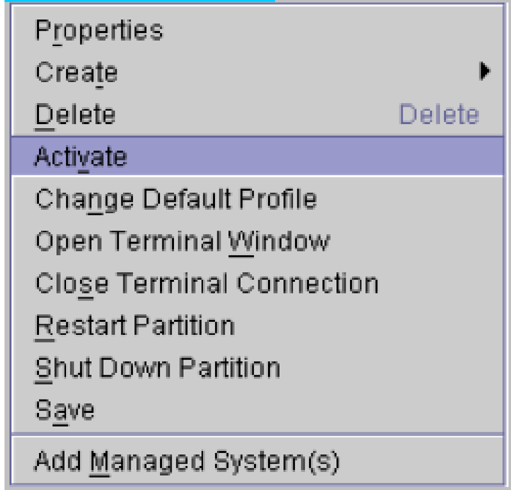
Fig. 11.



Name	State	Operator Panel Value
Server-9119-590-SN83...	Operating	
Partitions		
n05P01	Not Activated	00000000
n05P02	Running	
n05P03	Running	
n05P04	Running	
NIM	Running	
n05P05	Not Activated	00000000
VIOSA	Running	
Syst		

Note, that if you "right-click" any partition, you may select an option from the menu that appears, like "Restart Partition", or "Shutdown Partition" etc..
This is shown in figure 12.

Fig. 12.



Also note that you can choose "properties", with which you can change resources like number of PU, number of virtual cpu's, or adding an Adapter etc..

But, there still remains one important factor here: the ownership of IO adapters (fc, storage, network).

6.5 IO adapters and virtual adapters: VIOS and VIO commands:

Whether it's a VIOS, or an AIX LPAR etc.. Assigning resources as "min/desired/max PU" etc.., is not that hard to do,
and works the same way for those different types of LPARs.
In the former section, we created an AIX capable LPAR. There we "have seen" an example of assigning PU, virtual cpu's etc..

But in section 6.4, we left one thing out: selecting IO adapters.
Here are some choiches possible:
If it was an LPAR meant for AIX 5.2, you have to choose a dedicated storage adapter, and dedicated (private) network adapter.
If it was an LPAR meant for AIX *as of* 5.3, it can make use of virtualized scsi and virtualized ethernet, so
it does not need to have "private" adapters (although it may have them).

Note: Whether an AIX 5.3 or 6.1 LPAR uses "private" or "virtualized" (by VIOS) adapters, depends on certain things.
If you expect the througput to Storage to be very high, it's probably best to choose for private Fiberchannel cards.
Don't forget, if you use VIOS for Storage access, and/or network access, all the client LPARs share the capabilities
of the VIOS. Maybe that's OK for "avarage" Servers. Maybe it's NOT OK for high volume Servers. (*Well, it's only my opinion).*

If the VIOS LPAR is created, and the VIOS software installed "into" that LPAR, the VIOS can
be configured to provide virtual IO to it's client LPARs.

The "root" user of VIOS, is "padmin", so you set up a terminal to the VIOS and log on as padmin.

Per default, the VIOS will launch a limited shell. It is called the "ioscli".
You can break the limitations by running "oem_setup_env", to leave the restricted shell behind.

In VIOS, a large number of (new) commands are available (like lsvdev), as well as "normal" AIX commands (like lsdev),
which can use special flags, and produce output in the context of virtualization.

Let's suppose, that while creating the VIOS LPAR, we assigned (most or all) of the physical adapters
to that LPAR. Those adapters are related to Storage IO, and network adapters.
Thus, VIOS will own those physical adapters.
Thus, you have a couple of physical adapters assigned to the VIOS. One or more adapters
represent IO to (local or remote) Storage.

For simplicity, let's asume this Storage is a number of local disks (althoug it's not much different if it were remote storage).
Now, since all Storage adapters are placed at the VIOS, we have those disks visible in VIOS.
Now suppose hdisk2, hdisk3, and hdisk4, are still free (maybe hdisk0 and hdisk1 already belong to the VIOS software).

You want to assign those disks to the AIX LPARS N05LP01, N05LP02, and NO5LP03, *as "virtualized scsi"*.

Simple example of the above case:

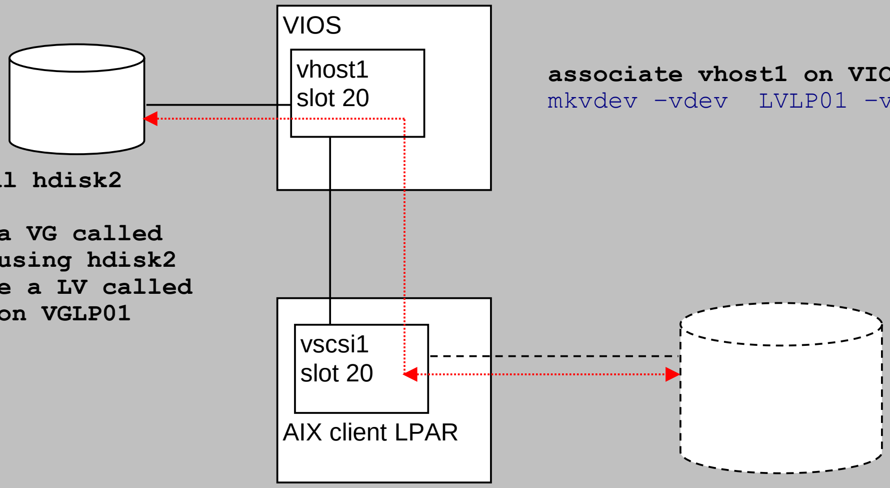
The VIOS is actually connected to the Storage, because it has assigned the IO adapters to it.
Suppose that on the VIOS, with the "lspv" command, we see that (as an example)
hdisk2, hdisk3 and hdisk4 are still unassigned.

Step 1:

On the VIOS, create a number of "Virtual SCSI Adapters". Say, you make 3 virtual adapters, to support 3 client LPARS,
who will use virtual Storage.
Now, you probably say: why that Virtual nonsense? Why create something as "virtual adapters" at the VIOS?
The Hypervisor will just use those "logical constructs", to be able to map a client "virtual adapter",

to a "VIOS virtual adapter. And if a relation exists between the real disk and such a "virtual adapter" at the VIOS, then client LPAR will "see" that storage as if it was it's own storage.
You will see that happen in a moment.

Fig. 13.



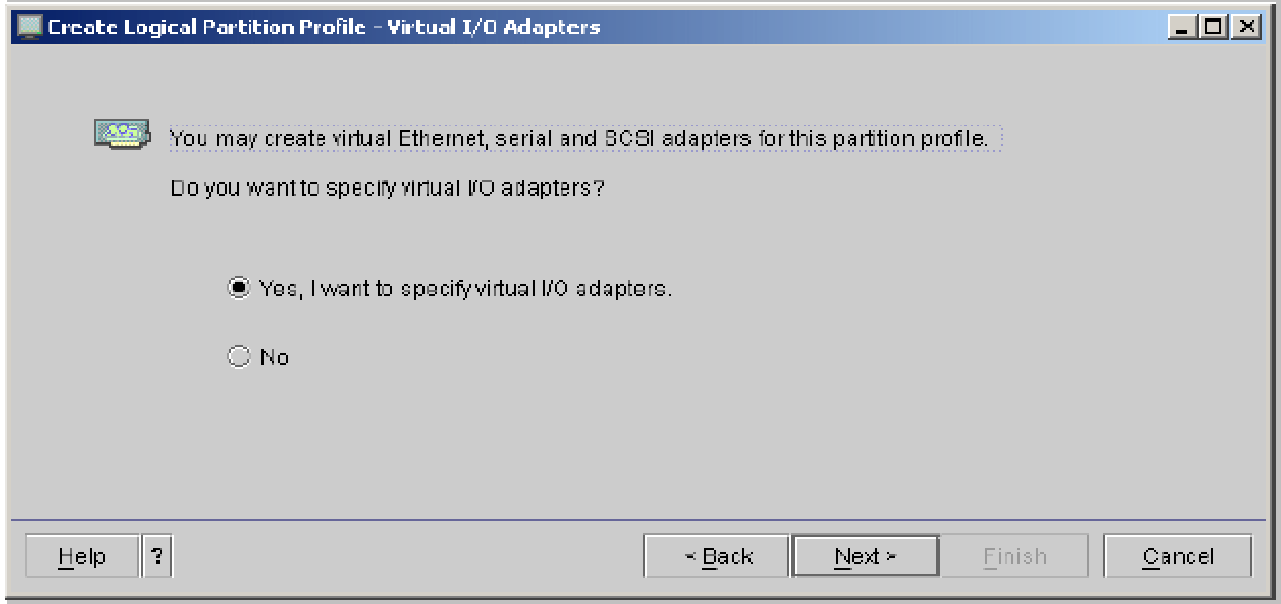
associate vhost1 on VIOS, with the LV on hdisk2

```
mkvdev -vdev  LVLP01 -vadapter vhost1
```

Now make a VG called "VGLP01" using hdisk2
Then, make a LV called "LVLP01" on VGLP01

So, just create the "virtual scsi adapters" in VIOS:

Fig. 14.



You may create virtual Ethernet, serial and SCSI adapters for this partition profile.

Do you want to specify virtual I/O adapters?

● Yes, I want to specify virtual I/O adapters.

○ No

You then are able to create a number of those adapters, called "vhost0", vhost1", "vhost2" etc.. And each of them uses a unique defined port or "slot", like 10, 20, 30 etc..

When done, you might check it from the VIOS command prompt:

```
VIOSA ~$ lsdev -virtual

name             status     description
vhost0           Available  Virtual SCSI Server Adapter
vhost1           Available  Virtual SCSI Server Adapter
vhost2           Available  Virtual SCSI Server Adapter
..
```

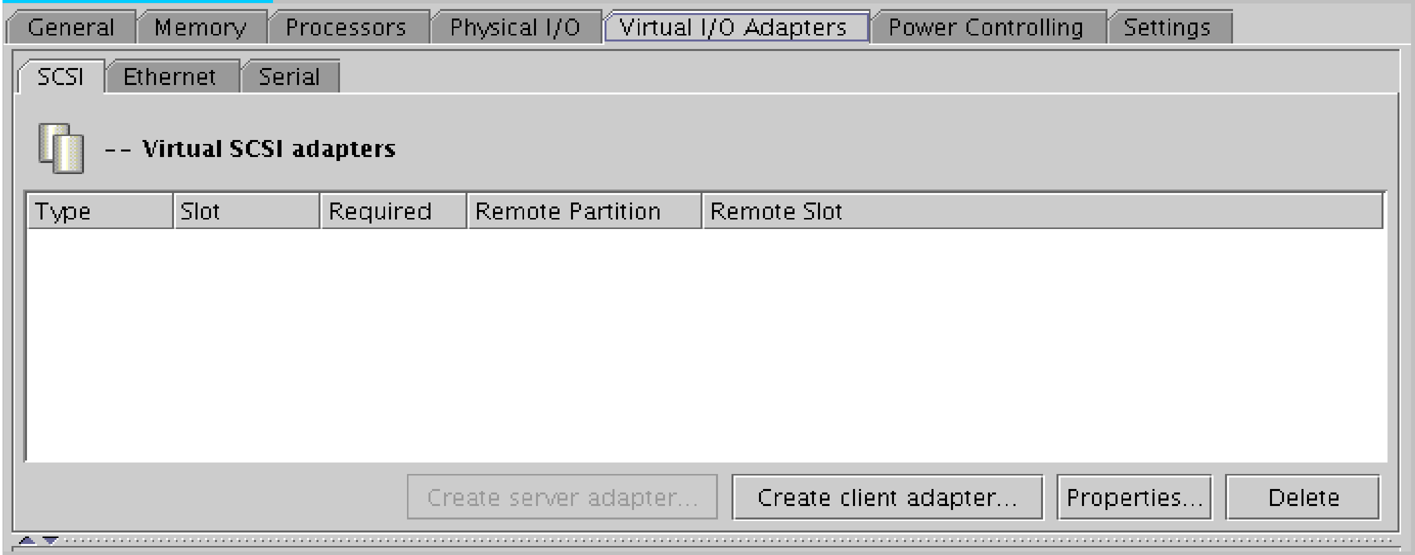
Here, on VIOS, you see the representations of those virtual controllers, by their names like "vhost0" etc.. Suppose thus, that you have created the following:

```
name               slot       purpose: to connect to a virtual client scsi adapter at the client
vhost0             20         N05LP01, using a client virtual scsi adapter in the same slot number
vhost1             30         N05LP02, using a client virtual scsi adapter in the same slot number
vhost2             40         N05LP03, using a client virtual scsi adapter in the same slot number
```

Step 2:

Create a "virtual scsi adapter" on the client, like LPAR "N05LP01".
On the HMC, rightclick the partition and choose "Properties" (like shown in figure 12).
In the screen that appears, choose "SCSI", then choose "Create client adapter".

Fig. 15.



Type	Slot	Required	Remote Partition	Remote Slot

Since we want to let this new client virtual scsi adapter on N05LP01, "vscsi0" to correspond to "vhost0 slot 20" on VIOS,
choose the following:
Client Slot: 20
Server Slot: 20

Step 3:

Return to the VIOS. Let's make "things" connect now.

Let's do something with the three disks that are visible from the VIOS.

```
VIOSA~$ mkvg -f -vg VGLP01  hdisk2            # So, we make a few Volume Groups.
VIOSA~$ mkvg -f -vg VGLP02  hdisk3
VIOSA~$ mkvg -f -vg VGLP03  hdisk4
```

Now (optionally), create LV's of a certain size. Virtual disks can either be whole physical disks or logical volumes.
The physical disks can either be local disks or SAN attached disks.

```
VIOSA~$ mklv -lv LVLP01 VGLP01 15G            # You would make the LV's only if you only want to expose
VIOSA~$ mklv -lv LVLP02 VGLP02 15G            # part of the physical disk to the client LPAR.
VIOSA~$ mklv -lv LVLP03 VGLP03 15G
```

The following statements actually will "associate" the LV's with the "virtual scsi adapters",
and will make the storage "known" to the client LPARS:

```
VIOSA~$ mkvdev -vdev LVLP01 -vadapter vhost0
VIOSA~$ mkvdev -vdev LVLP02 -vadapter vhost1
VIOSA~$ mkvdev -vdev LVLP03 -vadapter vhost2
```

So, for example, for LPAR N05LP01, the "virtual scsci adapter on slot 20" (vscsi0), is "bond" on the VIOS vhost adapter
on slot 20, which makes that the associated storage LVLP01 is accesible to the LPAR.

From now on, on the client LPARS, Physical Volumes like hdisk0, hdisk1 etc.. Are available.

6.6 Shared Ethernet Adapter on VIOS:

You *could* assign an LPAR it's own private ethernet adapter. But, just like virtual scsi, in a
LPAR with AIX version 5.3 or higher, we can create a "virtual ethernetadapter".

The VIOS owns all physical resources, but we will also create virtual ethernet adapters at the VIOS, and
we create a virtual ethernet adapter at each client LPAR. Again, those resources will be "mapped".

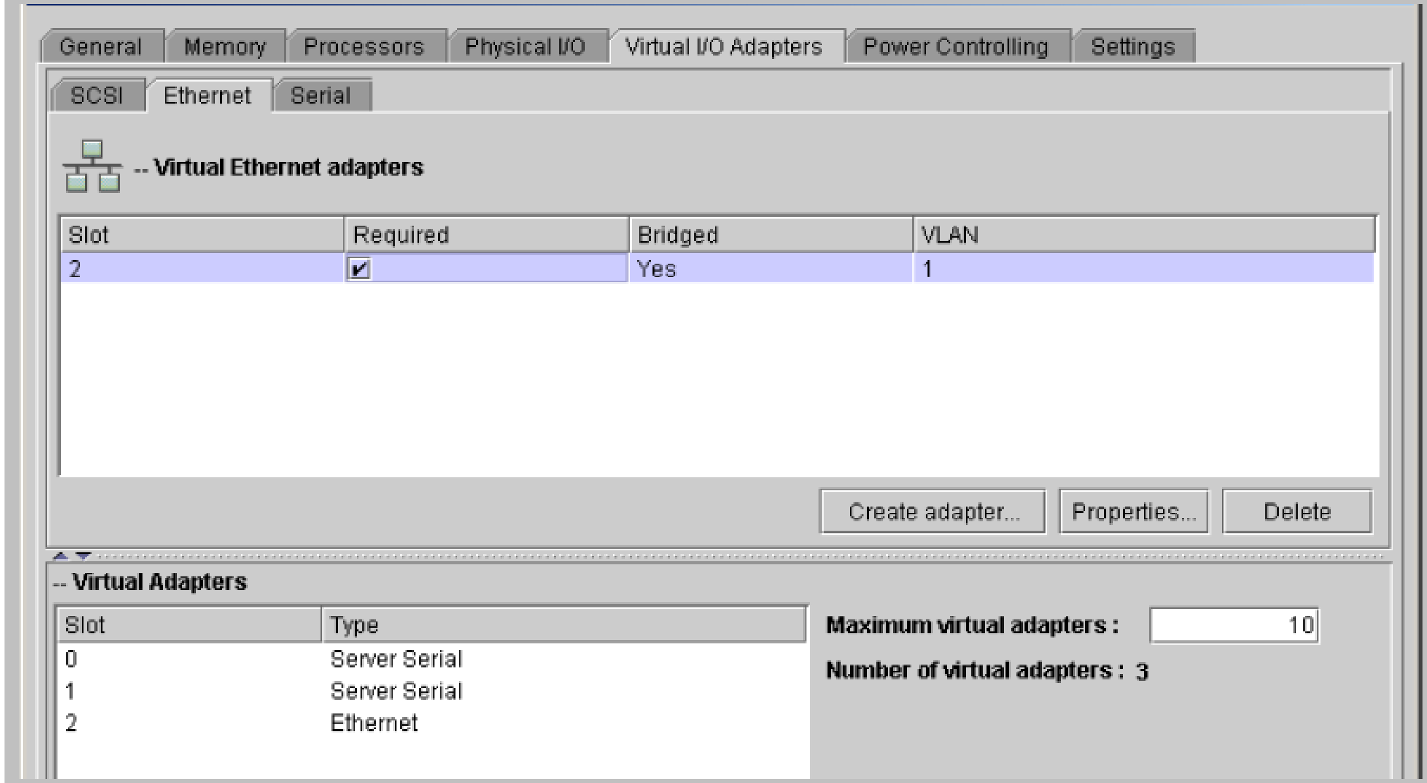
All LPARS should reside on the same IP subnet, like 10.10.10.0/24

If we want to let the LPARS communicate to the "outside world", 'through' the network capabilities of VIOS,
then at VIOS we have to create a "shared ethernet adapter".
All LPARS then have to reside into what is called "the same VLAN", and as such qualified by a unique VLAN ID.

So suppose we have the LPARS: LP01, LP02, and LP03.
Then on each LPAR, we create a virtual ethernet adapter.
Again, the "trick" is to let each chosen "slot" at a client virtual adapter, correspond to the slot

chosen at the VIOS.

So suppose we have this at the VIOS: Fig. 16.



-- Virtual Ethernet adapters

Slot	Required	Bridged	VLAN
2	✓	Yes	1

-- Virtual Adapters

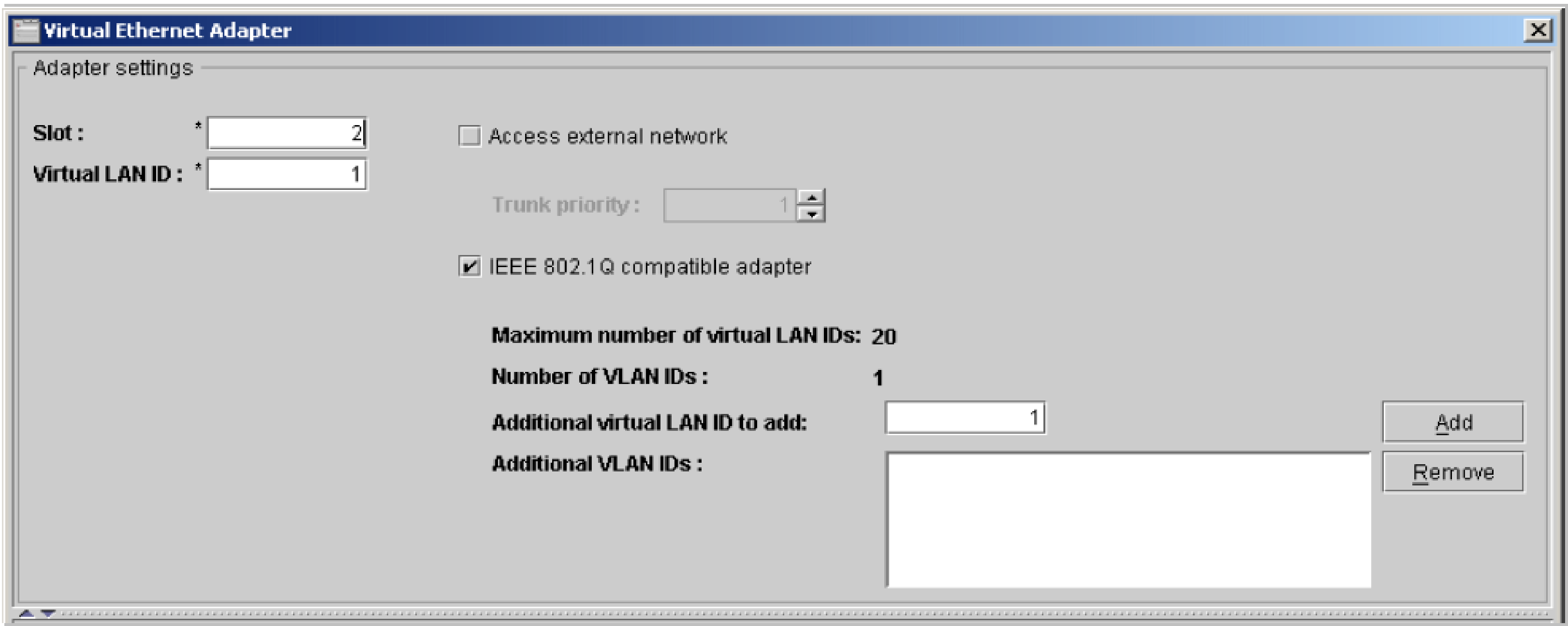
Slot	Type
0	Server Serial
1	Server Serial
2	Ethernet

Maximum virtual adapters : 10

Number of virtual adapters : 3

Thus, as you can see, we have one virtual ethernet adapter defined, residing at slot 2.

Now, at each client LPAR, we will create a virtual ethernet adapter, using the same slot and VLAN ID:



Slot : * 2

Virtual LAN ID : * 1

☐ Access external network

Trunk priority : 1

☑ IEEE 802.1Q compatible adapter

Maximum number of virtual LAN IDs: 20

Number of VLAN IDs : 1

Additional virtual LAN ID to add: 1

Additional VLAN IDs :

Fig. 17.

That's done, the mappings are allright now. But we need to let the LPARS communicate, through the VIOS,
to the "outside world" as well.
Such a mapping has not been done yet. So let's create a "shared ethernet adapter".

On the VIOS, we have this situation:

```
$ lsdev -type adapter

name        status              description
ent0        Available           2-Port 10/100/1000 Base-TX PCI-X Adapter (1410890
ent1        Available           2-Port 10/100/1000 Base-TX PCI-X Adapter (1410890
ent2        Available           Virtual I/O Ethernet Adapter (l-lan)
ide0        Defined             ATA/IDE Controller Device
sisscsia0   Available           PCI-X Dual Channel Ultra320 SCSI Adapter
vhost0      Available           Virtual SCSI Server Adapter
vhost1      Available           Virtual SCSI Server Adapter
vhost2      Available           Virtual SCSI Server Adapter
vhost3      Available           Virtual SCSI Server Adapter
```

So, presently at the VIOS, we have 2 real network adapters, and 1 virtual ethernet adapter.
Through curreseponding slotnumber (and VLAN ID) at the clients and VIOS, those resources are connected.
Now, let's create the (virtual) shared ethernet adapter (sea):

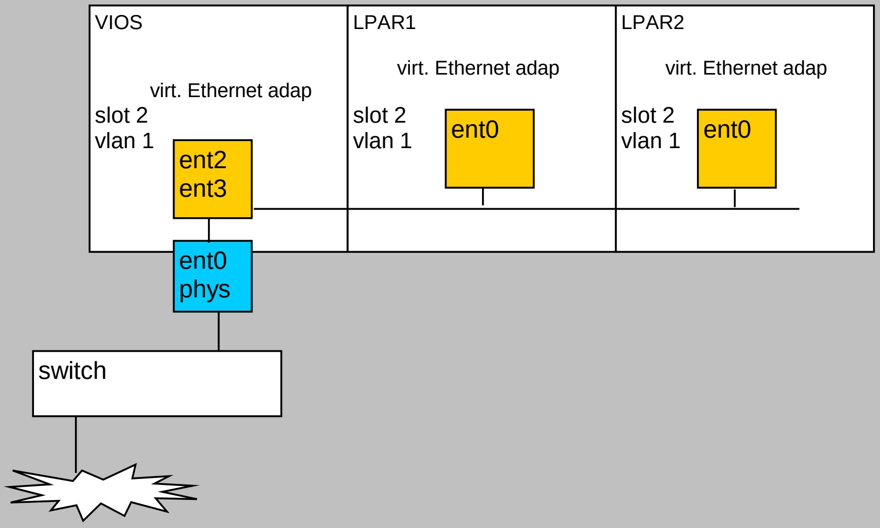
```
$ mkvdev -sea ent0 -vadapter ent2 -default ent2 -defaultid 1

ent3 Available
en3
et3
```

You do not need *to name* the sea, because the system will name it for you.
Here, "ent3" is our shared ethernet adapter. The client LPARS will use this one for communication
to the outside world.

The "mkvdev" command used here to create the "sea", is probably "hocus pocus" if you see it for the first time.

It says this: for the realization of the sea object, take the real adapter "ent0", and the virtual adapter "ent2"
(to which client LPARS have connected their virtual ethernet adapters, due to the equal slotnumber),
and take as a VLAN ID the number "1".

So, the sea called "ent3" is sort of mapped to the real adapter "ent0" (which is connected to the outside world),
and the virtual ethernet adapter "ent2", is coupled to each virtual ethernet adapter at the client LPARS.



6.7 Some other examples:

Example 1: The "lsmap" command on VIOS:

On VIOS, the "lsmap" command can be usefull.
It can show the relation between a "vhost" adapter and the "physical or logical" backing device (=disk or LV)
that is in use at the client LPARS.

Example:

```
VIOSA~$ lsmap -all -type disk lv

SVSA            Physloc                                          Client Partition ID
--------------- -------------------------------------------- ------------------
vhost0          U9117.570.65B61FE-V17-C11                        0x00000002

VTD                   lvlp01
LUN                   0x8100000000000000
Backing device        hdisk2
Physloc

SVSA            Physloc                                          Client Partition ID
--------------- -------------------------------------------- ------------------
vhost1          U9117.570.65B61FE-V17-C12                        0x00000003

VTD                   lvlp02
LUN                   0x8200000000000000
Backing device        hdisk3
Physloc

SVSA            Physloc                                          Client Partition ID
--------------- -------------------------------------------- ------------------
vhost2          U9117.570.65B61FE-V17-C18                        0x00000004

VTD                   lvlp03
LUN                   0x8300000000000000
Backing device        hdisk4
Physloc
```

Some output codes are:

SVSA	**Server Virtual SCSI Adaper**
Physloc	**Physical Location Code**
VTD	**Virtual Target Device**
LUN	**Logical Unit Number**
SVEA	**Server Virtual Ethernet Adapter**
SEA	**Shared Ethernet Adapter**
Backing device	**the pysical volume, or logical volume, that is attached to the virtual scsi adapter**

Example 2: Showing different types of storage on client LPARS:

How does it looks like when we want to show the Storage on client LPARS (like AIX clients), using different models:

>>> The following LPAR has storage solely from VIOS:

```
P550LP4:/root #  lspv
hdisk0          00cb61fe223c3926                    rootvg          active
hdisk1          00cb61fe2360b1b7                    rootvg          active
hdisk2          00cb61fe3339af9f                    appsvg          active
hdisk3          00cb61fe3339b066                    datavg          active

P550LP4:/root #  lsdev -Cc disk
```

```
hdisk0 Available  Virtual SCSI Disk Drive
hdisk1 Available  Virtual SCSI Disk Drive
hdisk2 Available  Virtual SCSI Disk Drive
hdisk3 Available  Virtual SCSI Disk Drive
```

>>> The following LPAR has storage from VIOS, but als has dedicated (private) FiberCards to a SAN:

```
P550LP5:/root #  lspv
hdisk0          00cb61fe09fe92bd                    rootvg          active
hdisk1          00cb61fe0a47a802                    rootvg          active
hdisk2          00cb61fe336bc95b                    appsvg          active
hdisk3          00cb61fe321664d1                    datavg          active

P550LP5:/root #  lsdev -Cc disk
hdisk0 Available          Virtual SCSI Disk Drive
hdisk1 Available          Virtual SCSI Disk Drive
hdisk2 Available 02-08-02 SAN Volume Controller MPIO Device
hdisk3 Available 02-08-02 SAN Volume Controller MPIO Device
```

So, here hdisk0, hdisk1 are from VIOS, but hdisk2, hdisk3 is from a SAN, to which P550LP5 is connected with local FC cards.

Chapter 7. Some keypoints on Managing Software.

7.1 Software Packaging in general:

Fileset:

A fileset is the smallest individually installable unit. It is a collection of files that provides a specific function. For example, the bos.net.tcp.client is a fileset in the bos.net package.

Package:

A package contains a group of filesets with a common function. This is a single installable image, for example, bos.net.

Licensed Program Product (LPP)

This is a complete software product, including all the packages and filesets required.

Bundle:

Sometimes, software is delivered as a "bundle", which is a collection of packages and/or filesets, that are needed to install a certain software product, or feature.

Commonly, the above products are installed by the "installp" command, or using smitty.

Naming convention:

-- In general, Packages, and Filesets (of various programs) follow the following standard naming convention.

```
LPP.<msg|lang>.package.fileset
```

where the <msg|lang> part may denote the "locale" or "language" the package or fileset is in.

-- The OS related filesets however, often are in the form:

```
bos.package.fileset.<version>.bff
```

like for example:

```
bos.adt.prof.6.1.1.2.bff
```

7.2 Operating System Levels, and Maintenance:

-- The Former way:

Formerly, an AIX version was determined by the Release and (Recommended) Maintenance Level (ML) like:

AIX 5.2 ML5

```
P520:/root #  oslevel -r
5200-05
```

-- The New way:

What is now (as of 2006) called "Technology Level", corresponds to what used to be called "Maintenance Level".

You can maintain your AIX operating system by installing Service Packs (SP), Program Temporary Fixes (PTF)
or Interim Fixes for the entire support life of the Technology Level.

Thus, between regular Technology Levels, IBM will release Service Packs,
which are PTFs that are grouped together.

Beside to install a SP ("grouped PTF's"), you can still install a single PTF, or a single Fix.

To see which Technology Level and Service Pack is currently installed, use "oslevel -s", like:

```
P550LP03:/root #  oslevel -s
5300-04-02
```

This is an LPAR with AIX 5.3, Technology Level 4, Service Pack 2.

(So, the ouput is: "Version - Technology Level - Service Pack").

In the new release strategy, one would for example speak about "AIX V5.3 TL6".

Service Packs are cumulative, so if service pack 4 is installed, all of the previous critical fixes from service packs 1 through 3 will also be installed.

A CSP, or "Concluding Service Pack" will identify the last service pack on a Technology Level.
In the "oslevel" output, this Service Pack will identify itself using the "CSP" string.

```
P570LP11:/root #  oslevel -qs
5300-05-03-0000
5300-05-02-0000
5300-05-01-0000             # Now, the "counting" of SP's starts with "1"
5300-04-CSP-0000            # TL 4, with the Concluding Service Pack
5300-04-03-0000             # TL 4, SP 3
5300-04-02-0000
5300-04-01-0000
5300-03-CSP-0000
```

If you would apply an individual PTF or Fix, the fourth part will get a number as well.

The new IBM methodology provides for (about) two TL releases per year.
SP's are generally released (about) every four to six weeks after the release of a new TL.

Overview OS Fixes and other replacements/maintenance:

Fix or APAR: APAR is short for Authorized Program Analysis Report. An APAR is a (usually small) emergency fix, or interim fix, to a unique problem on the system.
Usually, a fix of this type is installed using the "instfix" command.

PTF is short for Program Temporary Fix. A PTF is an updated fileset or a new fileset that fixes a previous system problem. PTFs are installed in the same way as regular filesets, that is, usually with smitty, or with the "installp" command.

A Service Pack is a number of PTFs that are grouped together.

A CSP, or "Concluding Service Pack" will identify the last service pack on a Technology Level.

A "Technology Level" update is a major update (formely called "recommended Maintenance Level").

Ways to maintain your OS:

There two ways to maintain your OS with respect to TL's, SP's and individual fixes:

- download it, and install it yourself using commands like installp, instfix, smitty.
- Using SUMA

7.3 AIX maintenance using SUMA:

SUMA is short for "Service Update Management Assistance".
It enables you to have your AIX system to automatically retreive TL's, SP's and even individual fixes.
It was first released with AIX v5.3.
It resembles "fixdist" somewhat, which could be found on older AIX versions.

You can let SUMA download a TL, a SP, or just a fix, at a date/time you want.
That is indeed a good thing.
Usually, you will use the "suma" command, and specify the options you want.

Per default, suma will save in: /usr/sys/inst.images

If you just let SUMA download TL's and SP's, then you are quite "safe".
But.. Nobody can deny that some specific software products are really tied to certain TL levels.
So, it is just not always feasable to "just" apply a SP or TL.
But you can at least let suma automatically download what you (might) need.
But there is a "but": you need a working connection to "Fix Central", and not everybody is able (or willing) to implement that.

View Basic SUMA options:

If you want to view "suma" right now, you can use the commandline, or the smitty tool,
which is the AIX "all in one" maintenance and configuration tool.

```
# smit suma
```

```
                       Service Update Management Assistant (SUMA)

Move cursor to desired item and press Enter.

  Download Updates Now (Easy)
  Custom/Automated Downloads (Advanced)
  Configure Suma
```

```
                       Download Updates Now (Easy)

Move cursor to desired item and press Enter.

  Download by APAR Number
  Download by Fix Type
  Download Maintenance Level or Technology Level
  Download Service Pack
  Download All Latest Fixes
  Download by Fileset Name
```

To view basic suma settings from the commandline:

```
# suma -c                      # Lists the SUMA global configuration settings
# suma -D                      # Lists the SUMA task defaults
# suma -l <TaskID>             # Lists the specifics of the task with that TaskID
# suma -l                      # Lists all tasks
```

Implementing SUMA:

As you can see from the figures above, smitty has a fairly simple interface to suma.
But knowing how to basically configure suma from the commandline, is required as well.
That is, reckognize the general form, and understand what a command does (by viewing a few examples here).

The basic form of the 'suma' command is:

```
suma { { [ -x ] [-w ] } | -s CronSched } [ -a Field=Value ]... [ TaskID ]
```

```
-x:                  execute now
-w:                  writes or saves a SUMA task
-s:                  schedules a SUMA task, using a crontab like format
-a Field=Value       many options can be specified in your command, or scheduled suma task.
                     Some of the most important are:

                     RqType:
                     RqType= APAR | PTF | ML | TL | SP | Fileset | Security | Critical | Latest
                     When suma is run with an RqType of Security, Critical, or Latest, the RqType is the only required field.

                     RqName:
                     Here you specify the name of the item that you request, like IY12345, or bos.rte.lvm etc..

                     Action:
                     Can be "Preview", "Download", or "Clean".
                     Preview: only a preview of the download
                     Download: download in the default location, or what is specified in DLTarget (DownLoad Target).
                     Clean: Download, and cleanup of all files that are not needed anymore.
```

Here are a few examples:

```
# suma -x -a Action=Preview  -a RqType=Latest                    # This will be executed now.
# suma -x -a Action=Download -a RqType=Latest                    # This will be executed now.
# suma -s "30 3 15 * *"  -a RqType=Security \                    # a scheduled command that will download Security fixes
      -a DisplayName="Download fixes on each day=15"             # at 03.30h on every day_of_month=15
```

You see? The suma commands are actually fairly easy to read (and to create).

7.4 Applied and Committed Software:

Generally speaking: software can be "applied" or "committed" (and if it was applied,
then afterwards it can be "rejected").
For OS related filesets and packages, this is certainly true.
However, since "a certain command" can be recommended by a Manufacturer (or it says so in a readme),
you might use a certain command that also "commits" the software right away, after you have entered that command.

But in general, when a update is installed or applied, it enters the applied state and
becomes the currently active version of the software.
In this case, the previous version of the update is stored in
a special save directory "/usr/lpp/PackageName". This allows you to restore the previous version,
without having to reinstall it.

Software that has been applied to the system can either be committed (that is, make permanent), or
rejected. There are several ways to do that.
For example, The "installp -s" command can be used to get a list of applied products
and updates, that are available to be either committed or rejected.

7.5 Some important commands in managing software:

Some important commands are:

```
- instfix         Used to install APARs (fixes)
- lslpp           Used to query for filesets
- installp        This is the "general" install command
- inutoc          Updates the "table of contents"
- geninstall      Generic installer that can handle various package formats.
- smitty          The "friendly all-in-one" menu, also usable for installing software
- lppmgr          Manages sources in the lpp_source in the NIM environment.
- lppchk          Used for verifying a correct installation.
```

Now, there is not really a fixed, required location, where you must place the fix, or the filesets.
You could for example, create a directory "/tmp/inst" where you place the filesets in.

Let's take a look at a few examples.

smitty:

If you just start "smitty", you will see how easy it it to install software. It's just one of
the first menu entries: "Software Installation and Maintenance".

instfix:

Simplyfied command:

```
instfix -k [-i] <apar> -d <device>
```

Examples:

```
# instfix -k IX75893 -d /dev/cd0     # install the fix from the CD or DVD /dev/cd0
# instfix -k IX75893 -d .            # install the fix from the current directory
# instfix -T -d /dev/cd0             # To list fixes that are on a CD-ROM in /dev/cd0
# instfix -ik IX12345                # Is this fix already installed?
Not all filesets for IX75893 were found.
```

Or use the "shortcut" smitty menu

```
# smitty instfix
```

```
                          Update Software by Fix (APAR)

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* INPUT device / directory for software              []                                                   +
```

lslpp:

The lslpp command displays information about installed filesets or fileset updates.
It can use many flags. Some of the more important ones are:

-l: Displays the name, level, state and description of the fileset.
-h: Displays the installation and update history for the fileset.
-p: Displays requisite information for the fileset.
-d: Displays dependent information for the fileset.
-f: Displays the filenames added to the system during installation of the fileset.
-w: Lists the fileset that owns a file or files.
-R: specifies a user designated install directory.
-L: Displays extended listing.

Full syntax:

```
lslpp [-R { path | ALL } ] { -d | -E | -f | -h | -i | -l | -L | -p } [ -a] [ -c] [ -J ] [ -q ] [ -I ] [ -O { [ r ] [ s ]
                   [ u ] } ] [ FilesetName ... | -b File | all ]

lslpp [-R { path | ALL } ] -w [ -c ] [ -q ] [ -O { [ r ] [ s ] [ u ] } ] [ FileName ... | all ]

lslpp [-R { path | ALL } ] -L -c [ -v]

lslpp [-R { path | ALL } ] -S [A|O]

lslpp [-R { path | ALL } ] -e
```

Examples:

Sometimes you need to find out whether one or more filesets are present on your system, for example
as a prerequisite for some other software. Then use "lslpp -h" as in the following example:

```
# lslpp -h bos.adt.include bos.adt.l1b bos.adt.l1bm \
           bos.net.ncs 1for_ls.compat 1for_ls.base
```

In order to display the name, level, state of the "bos.adt.include" fileset, use

```
# lslpp -l bos.adt.include

  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.adt.include           5.3.0.37  COMMITTED  Base Application Development
                                                 Include Files
```

In that command, you can use "wildcards" as well, as in:

```
# lslpp -l "bos.adt.*"
```

In order to display all files in the inventory database which are related to the vmstat command, use

```
# lslpp -w "*vmstat*"
```

You can create "long" listings with lslpp, which you can ofcourse "grep" on the information you want, like:

```
# lslpp -l | grep -i Java
```

As explained in Chapter 1, the lslpp will query *that* part of the ODM database, which is related to filesets.
See chapter 1.

inutoc:

Creates a .toc file (table of contents file) for directories that have "installable" files or filesets.

Usually, you do not need to use inutoc. For example, if you have media from some manufacturer,
then you can just go ahead with the installation (with, for example, using smitty).
But suppose you have "just a number" of filesets, in some directory, and you know that you can install those,
then create a "table of contents" file first.

Per default, inutoc operates on the "/usr/sys/inst.images" directory.
But you can use inutoc on every location, by just specifying that location, like for example:

```
# inutoc /tmp/install
```

installp:

The useage of the installp command is often an exam subject.
This command has nummerous flags. You should know the most important flags used with this command.

This command can ofcourse be run from the commandline, but many "smitty" software menu's will use it too.

```
# installp -r                 # to reject applied updates.
# installp -a                 # applies software.
# installp -ac                # applies and commit software.
# installp -C                 # cleans up after an interrupted installation and removes all software pieces.
# installp -d <device>        # used to specify the device where to install from, like "/dev/cd0".
# installp -F                 # forces the installalation, even if the same version already exists.
# installp -l -d <device>     # list all installable software on that device.
# installp -L -d <device>     # displays the contents of the media by looking at the table of contents (TOC).
# installp -X                 # expand filesystems as necessary.
# installp -e                 # logs to a logfile
# installp -q                 # Specifies quiet mode, which suppresses the prompt for the device, except media volume change.
# installp -Y                 # agrees to software license requirements. Only valid using the -a flag.
# installp -u                 # Removes the specified software from the system.
# installp -s                 # Gets a list of all applied software that can be committed or rejected.
# installp -v                 # verifies the checksums after install. it can be an extra saveguard with a remote install.
# installp -g                 # when used to install or commit, this flag automatically installs or commits,
                                respectively, any software products or updates that are requisites of the specified software
                                product.  When used to remove or reject software, this flag automatically removes or rejects
                                dependents of the specified software. The -g flag is not valid when used with the -F flag.
```

Examples:

To list all software products and all installable options on the cd0 device, use:
```
# installp -L -d /dev/cd0
```

To commits all applied LPPs (or PTFs etc..), use:
```
# installp -c -g -X all
```

The following installs those three filesets from cdrom:
```
# installp -qaX -d /dev/cd0 bos.sysmgt.nim.master bos.sysmgt.nim.client bos.sysmgt.nim.spot
```

The following example installs all available filesets in the "/cdrom/usr/sys/inst.images" source directory
to the default locations, as specified in Filesets, and writes an installation log file to /tmp/install.log.
```
# installp -aXYgd /cdrom/usr/sys/inst.images -e /tmp/install.log all
```

geninstall:

This is a generic installer that can install software products of various packaging formats, like
installp, RPM, SI, and ISMP.
Beginning in AIX 5L, you can not only install installp formatted packages, but also RPM and
Install Shield Mutli-Platform (ISMP) formatted packages. Use the Web-based System Manager,
SMIT, or the geninstall command to install and uninstall these types of packages.
The geninstall command is designed to detect the format type of a specified package and run the
appropriate install command.

Example:

```
# geninstall -d /dev/cd0 all
```

If you using the geninstall command to install RPM or ISMP packages, use the prefix type to designate
to the geninstall command the type of package you are installing. In AIX 5L, the package prefix types
are the following:

I: installp format
R: RPM format
J: ISMP format

For example, to install the cdrecord RPM package and the bos.games installp package, type the following:

```
# geninstall -d/dev/cd0 R:cdrecord I:bos.games
```

lppmgr: (specifically used at NIM)

Manages software in the "lpp_source" on the NIM master.

lppmgr is designed to perform the following functions on an existing installp image source
(also known as an lpp_source in the NIM environment):
Important flags:

```
-d: Directory or Device.
-u: Remove duplicate updates.
-b: Remove duplicate base levels.
    Thus eliminating updates which are the same level as bases of the same fileset. Such updates can create conflicts
    that lead to installation failure.
-k: Remove message and locale filesets other than the language you specify.
-x: Remove superseded filesets.
-r: Actually removes filesets, because the default will only lists what will be removed.
-X: Remove non-system images from a NIM lpp_source resource.
```

Example:

```
# /usr/lib/instl/lppmgr -d /export/nim/6100/lpp_source  -u -b -x -r
```

```
>> this will remove
-u = removes duplicate update images
-b = removes duplicate base images
-x = removes superceeded updates
-r = actually does the removal. If the ‘-r’ flag is not included, the lppmgr command will run in “preview mode”.
```

lppchk:

This command checks the current files of a program, or a filelist, against the
Software Vital Product Data (SWVPD) of the ODM (see also Chapter 1).
It will check for file sizes and checksum values, or symbolic links.

Most important flags:

```
-c                Performs a checksum operation on the FileList items and verifies that the checksum and the file size
```

```
                  are consistent with the SWVPD database.
-l                Verifies symbolic links for files as specified in the SWVPD database.
-v                Does an extensive integrety check. All software products are registered in the ODM as consisting
                  of a root part, and a usr part. These SWVPD entries are crosschecked along with the existence
                  of the actual files on the filesystems.
```

Examples:

Complete integrety check:

```
# lppchk -v
```

Check a package on checksums and filesizes:

```
# lppchk -c X11.fnt
```

Chapter 8. Some keypoints on WPARs.

Please be aware that this chapter describes an AIX v.6 feature only.
This chapter contains really "minimal" information on WPARs, but it's likely
that it's all you need to know for the exam 223.

8.1 Conceptual Overview WPAR:

A WPAR is a software based 'entity', running within a AIX v.6 based LPAR.
A (system) WPAR behaves like an autonomous "virtual AIX machine", running within an AIX v.6 LPAR.

Since it is entirely software based, it has no relationship with the Power architecture, or firmware etc..,
and you do not need a HMC. It's a feature of AIX v6 only.
You can create, list, manage, start/stop, and remove them using the commandline, or smitty, and other tools.

WPAR is short for "Workload Partition". You may regard it as a sort of "mini AIX", running
within your LPAR. And, you might even create a number of autonomous WPARs within one AIX v6 based LPAR.

It is just as if you are "subsetting" your LPAR into a number of autonomous systems.

If you would have, say 5 independend WPARS running in a single LPAR, and *if* that LPAR suddenly
experiences a "serious problem", then all of your WPARS have a problem too.
That’s a bit of a dramatic scenario ofcourse, but it nicely illustrates how these "virtual machines"
are tied into that single LPAR.
But the following could be quite realistic: system maintenance on the LPAR can affect all WPARs.

Note:
However, a WPAR can be moved to another LPAR, where only a relatively short "freeze" time
is involved (called Application Mobility).
If you want to use this feature, the r/w filesystems of the WPAR must reside on a NFS Server.

Now, a physical machine like a p570 can be partitioned into several LPARS. In a AIX v6 based LPAR, you
can thus create WPARS, and thereby having even a higher degree of "partitioning".
If you want, you may go back to figure 4 of chapter 6. This figure "puts things into perspective".

In figure 4, you see an AIX v6 based LPAR, where a couple of WPARs are running in.

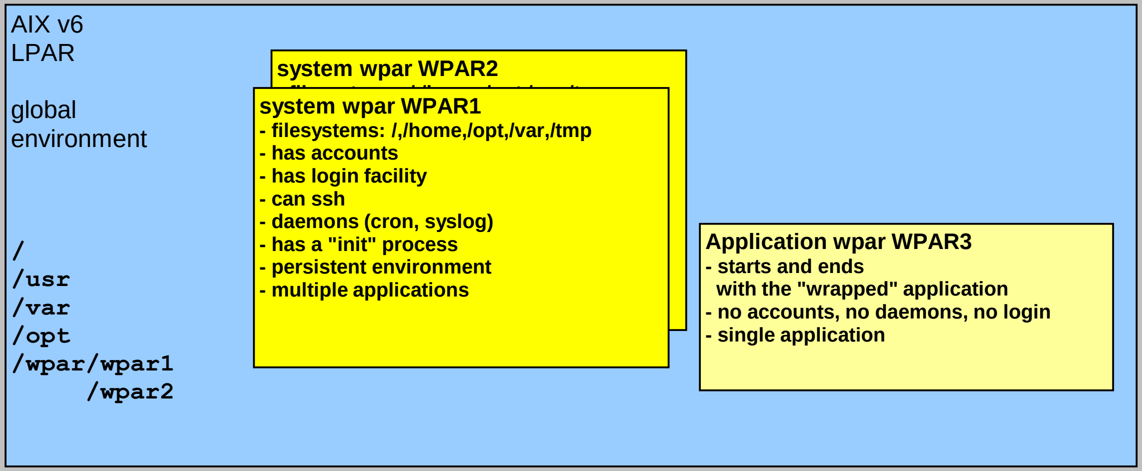
There a two types of WPARS:

- System WPAR: This resembles a "mini" AIX environment, with it's own r/w mounted filesystems, and daemons like cron. Furthermore, it has it's own users and groups, and network features. A system WPAR is persistent, until you stop it using the 'stopwpar' command.
- Application WPAR: This one is much more limited, and can be viewed as a 'wrapper' around a process, thereby isolating it further. An Application WPAR starts and ends, when the app starts and ends.

Since the WPARs are running in a single LPAR, they use the resources of that LPAR.

In the context of WPARs, the parent LPAR is called the "global environment".
The following figure hopefully illustrates the relation of a parent LPAR and it's WPARs.

Fig. 18.



A system WPAR thus looks like a mini AIX environment. It has mounted file systems, private accounts, it's own set of daemons, and even an "init" process (!), but it does not have it's own "kernel" or own devices. Those are inhereted from the "parent LPAR".

In the following sections, whenever we talk about "global", like "global filesystems", it's meant to be the filesystems from the parent LPAR (thus the global environment).

WPARs are certainly "great" for test- and development "streets". For production, you should be carefull.
A WPAR is dependent to the availability of the Hosting LPAR, although "Application Mobility" options do exist.

A WPAR is also about "Resource Control":

It's very easy to create a WPAR using the mkwpar command.
Here is an example:

```
570LP02:/root # mkwpar -n wpar1 -h wpar1 -N netmask=255.255.255.0 address=10.10.10.20 -r
```

This create as system WPAR, with an administrative name of "wpar1", and a hostname of "wpar1", and some network attributes are set as well.

Now let's start wpar1:

```
570LP02:/root # startwpar wpar1
```

If you want to login into this OS container, use the "clogin" command:

```
570LP02:/root # clogin wpar1
```

But since a WPAR is short for "Workload Partition", it should have a relation to setting "Workload Boundaries" (in the context of Resource Control) as well.
Thus, we can expect that this sort of flags exists in the "mkwpar" command as well, like "CPU=x%-y%", meaning a lower and upper bound of that resource (taken from the global environment).

Control at LPAR boottime:

The /etc/rc.wpars command invokes the startwpar command on all workload partitions with the autostart option.
This means that the WPAR was created using the "-A" flag with the mkwpar command, and thus means they are "auto started" when the host OS (the AIX global enviroments) starts up.

8.2 Creating and managing WPARS:

8.2.1 Creating, and listing, WPARs:

To create a WPAR, you can use smitty, or the "mkwpar" command, or the webbased "WPAR Manager".

Using smitty:

```
570LP02:/root # smitty wpar
```

Or using the commandline:

```
570LP02:/root # mkwpar wpar1
```

That command would create a wpar, called "wpar1", with all the defaults that are associated with the mkwpar flags.
Most often, that is not what we want, so here are a few more realistic examples:

```
570LP02:/root # mkwpar -n wpar1 -h wpar1 -N netmask=255.255.255.0 address=10.10.10.20 -r
570LP02:/root # mkwpar -n wpar2 -h wpar2 -R active=yes CPU=10%-20%,50% totalProcesses=300
```

In the second example, you can see that we have used command options that limit the resource usage, like CPU, and the maximum number of processes.

A (very) simplyfied version of the mkwpar command is:

```
mkwpar -n <wparname> -h <hostname> -d <base directory> -N <network attributes> -l -R attribute=value -A -s

-l makes a private copy of the global /opt and /usr and thus, makes them (in principle) writable to WPAR users.
-n designates the administrative WPAR name.
-h designates the "hostname" of this "mini AIX" system.
-d specifies a "base directory" for this workload partition. If you do not specify a directory name, /wpars/<wparname> is used.
   This is where "/home", "/tmp" etc.., are stored.
-A determines if an entry is recorded into the "/etc/rc.wpars" file (for autostart purpose).
-R attribute=value

With the "-R attribute=value", especially "Resource Control" parameters can be given on the command line.
For example:
CPU=m%-SM%,HM%                      Specifies the percentage processor limits for the WPAR.
memory=m%-SM%,HM%                   Specifies the percentage memory limits for the processes of the workload partition.
totalVirtMem=n[M|MB|G|GB|T|TB]      Specifies the total virtual memory for the WPAR.
```
many more attributes are available.

Remarks about Filesystems:

By default, the file systems for a new system WPAR are located in the /wpars/wpar_name directory.
If you create a system WPAR (with the mkwpar) command, you would see output about creating and mounting (private) filesystems, as well as copying fases of a number of AIX related filesets into the private directories.
With the "-d" flag in the "mkwpar" command, you can assign another "base directory" instead of the default of "/wpar".

Per default (not using the -l flag), the /usr and /opt file systems of the workload partition are mounted over the global /usr and /opt file systems in read-only mode. These system related filesystems, are thus "shared".
But if you use the "-l' flag, then a private copy of the global /opt and /usr are created (using more diskspace).

An Example of creating wpars, and listing them:

Let's create a couple of WPARs, login, and list them from the global environment.

>> Create WPARs:

```
570LP02:/root # mkwpar -n wpar1 -h wpar1 -N netmask=255.255.255.0 address=10.10.10.21 -r

mkwpar: Creating file systems...
 /
 /home
 /opt
 /proc
 /tmp
 /usr
 /var

<< End of Success Section >>

FILESET STATISTICS
------------------
  241  Selected to be installed, of which:
      241  Passed pre-installation verification
  ----
  241  Total to be installed

+-----------------------------------------------------------------------------+
                         Installing Software...
+-----------------------------------------------------------------------------+

Filesets processed:  6 of 241  (Total time:  2 secs).

installp:  APPLYING software for:
        X11.base.smt 6.1.0.1
```

etc.. (much output omitted)

In the same way, we create WPAR "wpar2".

>> Start the WPARs:

Let's now start those 2 WPARs:

```
570LP02:/root # startwpar wpar1
570LP02:/root # startwpar wpar2
```

>> View the WPARs from the "global environment" (the LPAR):

From the Host OS, that is, the global environment (the LPAR), use the "lswpar" command
to view the state of the WPARs:

```
570LP02:/root #  lswpar

Name        State  Type  Hostname          Directory
-----------------------------------------------------
wpar1       A      S     wpar1             /wpars/wpar1
wpar2       A      S     wpar2             /wpars/wpar2

570LP02:/root #
```

As you can see from this output, both WPARs are "active" (State=A), and they are "system WPARs (Type=S)".
Also, the base directories are "/wpars/<wpar_name>"

>> Login in a WPAR:

let's login to "wpar1"

```
570LP02:/root # clogin wpar1

*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 6.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************

#
```

Many familiar commands like "ps -ef", "df" etc.., can be entered and those commands
only "applies" to that WPAR.

8.2.2 Starting and stopping a WPAR:

-- Start / Stop an individual WPAR.

You can star/stop a WPAR from the global environment using the startwpar/stopwpar commands.

To start a system WPAR, run the following command in the global environment:

```
# startwpar wpar_name
```

For example:

```
570LP02:/root # startwpar wpar1
```

To start a system WPAR in maintenance mode, run the following command in the global environment:

```
# startwpar -m wpar_name
```

This starts the WPAR with all functionality, except for networking, so external user access is not possible.

To stop a WPAR, use the stopwpar command, as in the following example:

```
570LP02:/root # stopwpar wpar1
```

To "halt" or "abort" a WPAR, use the -F flag as in:

```
570LP02:/root # stopwpar -F wpar1
```

-- Global setting: starting WPARs when the LPAR boots:

If you use the "-A" flag in the mkwpar command, an entry is added in the "/etc/rc.wpars" file.
This file determines which wpars should auto start when the LPAR is booted.

The "/etc/inittab" file (which determines part of the bootprocess) contains a record to execute the "rc.wpars" file.

Chapter 9. Some keypoints on boot and shutdown.

9.1 Activating an LPAR from the HMC, and boot "normally", or boot to "SMS":

You can activate an LPAR, and boot the OS, from the graphical interface available on the HMC.

Usually, you will let the LPAR boot "normally", but sometimes you want to enter the SMS (System Management Service) menu.

Suppose a partition is in the "Not Activated" state.
From the HMC, rightclick the partition and choose "Activate".

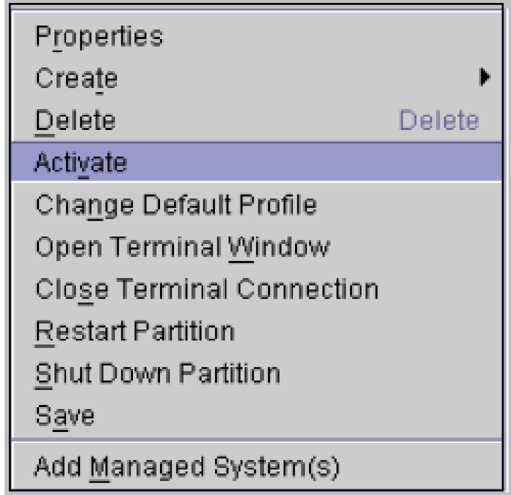


Fig. 19.

Next, you are able to select a 'profile'. At the very least, the default profile for this LPAR exists.
If more profiles exist, they are shown here, and you can select the one you want to use right now.
In the same screen, check the "Open a terminal window or console session". After that, click the "Advanced" button.

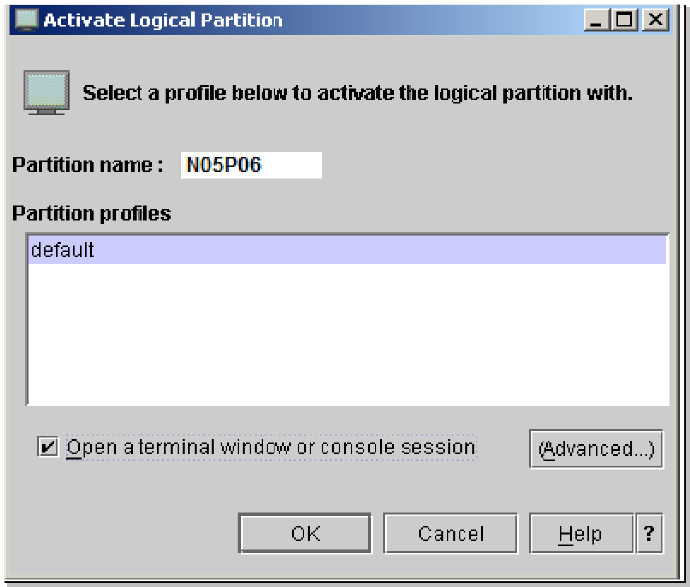

Activate Logical Partition
Select a profile below to activate the logical partition with.
Partition name : N05P06
Partition profiles
default
Open a terminal window or console session
Advanced...
OK
Cancel
Help
?

Fig. 20.

After clicking "Advanced", you can see that you can choose a "Bootmode".

- The most common option here, is "Normal", which normally would be used. It will just start the OS that's installed in the LPAR.
- Another important option is, that you can boot into the SMS menu. That menu will enable you (among other things) to perform maintenance (like recovery of rootvg).

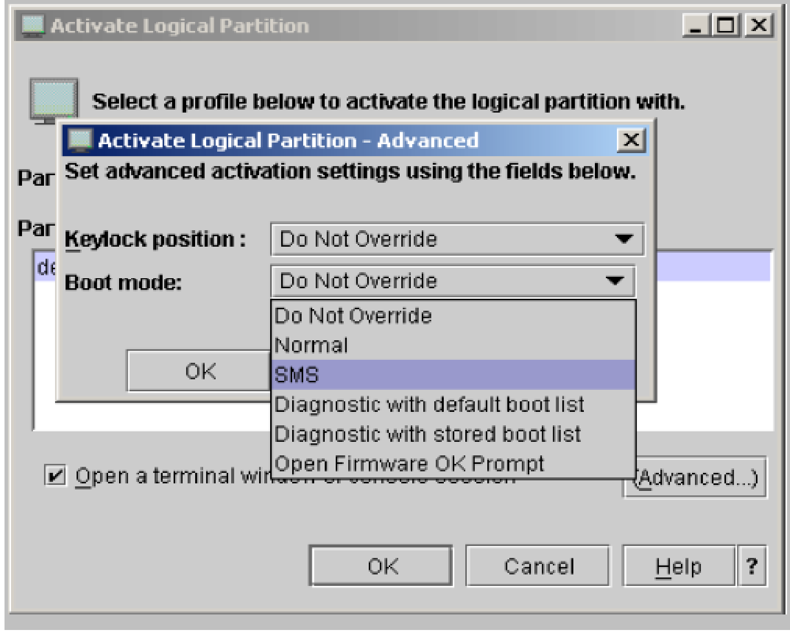

Activate Logical Partition
Select a profile below to activate the logical partition with.
Activate Logical Partition - Advanced
Set advanced activation settings using the fields below.
Keylock position : Do Not Override
Boot mode: Do Not Override
Do Not Override
Normal
SMS
Diagnostic with default boot list
Diagnostic with stored boot list
Open Firmware OK Prompt
OK
Open a terminal window or console session
Advanced...
OK
Cancel
Help
?

Fig. 21.

In the figure above, suppose we choose the SMS menu. After a short while, the SMS menu appears:

```
Main Menu:

  1. Select Language
  2. Setup Remote IPL (Initial Program Load)
  3. Change SCSI Settings
  4. Select Console
  5. Select Boot options

Type the number of the menu item and press Enter:
```

Let's demonstrate the use of this bootoption.
Suppose you want to boot from the AIX installation DVD.

Then, from the above SMS menu, choose 5 "Select Boot Options".
Then, press the 1 key and press Enter to select 1. "Select 1st Boot Device".
Ten press the 3 key and press Enter to select option 3, which is the (local) CD/DVD device.

Booting from DVD: How to start the installation of AIX, or go to Sytem Recovery options.

Let's alreay take a look how we could install AIX, or enter the "recovery options", if we would boot from
the AIX installation DVD (Volume 1).

Browse the above SMS menu system a bit more, until you are able to make the CD/DVD device the first bootable device.

After the next boot (from the AIX DVD Volume 1), you will see the following menu:

```
            Welcome to Base Operating System
            Installation and Maintenance

Type the number of your choice and press Enter. Choice is indicated by >>>

   1 Start Install Now with Default Settings

   2 Change/Show Installation Settings and Install

   3 Start Maintenance Mode for System Recovery

   4 Configure Network Disks (iSCSI)

   88 Help ?
   99 Previous Menu
>>> Choice [1]: 3
```

As you can see, here you can choose to install the OS, or you can choose "Start Maintenance Mode for System Recovery".
Please refer to the next chapter for more information. This chapter only deals with bootoptions.

9.2 HMC commands for Start/Stop, or altering boot behaviour of an LPAR:

See also section 6.3, for an explanation of the relation of the HMC with an LPAR.

Alongside the graphical activation options, using the HMC, you can also use the commandline from the HMC.
Here, you can find a few examples:

1. To shutdown a certain LPAR on a managed system, you can use a command similar to this example:

```
hscroot@hmc-op:~> chsysstate -r lpar -m pSRV570 -o shutdown -n N05LP04
```

2. To boot a certain LPAR on a managed system, you can use a command similar to this example::

```
hscroot@hmc-op:~> chsysstate -r lpar -m pSRV570 -o on -n N05LP04 -f profile
```

3. To perform a partition boot into SMS you can use a command similar to this example::

```
hscroot@hmc-op:~> chsysstate -r lpar -m pSRV570 -o on -n N05LP04 -f profile -b sms
```

4. To perform a partition boot into open firmware OK prompt, use a command similar to this example:

```
hscroot@hmc-op:~> chsysstate -r lpar -m pSRV570 -o on -n N05LP04 -f profile -b of
```

In these examples:
- the managed 'system p' system is "pSRV570"
- the LPAR is "N05LP04"

(Quite some more boot/shutdown HMC commands exists).

9.3 AIX "boot related" commands:

Next, we will discuss (in a lightweight fashion) some important AIX boot related commands.

9.3.1 The bootlist command:

The "bootlist" command (used from the AIX commandline) let's us view, or change, the list of bootable devices.
If the system just boots from local private devices (like a local hdisk0), the below theory is probably allright.

IMPORTANT: depending on the system model, other constraints may exist for a local disk to be bootable.

And, in modern complex systems with the use of VIOS and virtualized scsi, with or without a SAN, or other complex storage, the below theory could fail. It just depends on the configuration.
In many cases, the first admins who have setup the systems, have found out "the hard way", how it really works in that particular environment. Anyway, the basic theory is this:

At boottime, once the POST is completed, the system will search the boot list for a bootable image. The system will attempt to boot from the first entry in the bootlist.
Its always a good idea to see what the OS thinks are the bootable devices and the order of what the OS thinks it should use.

The purpose is ofcourse, to find a "boot logical volume" (blv), to boot the system (usually the LPAR) from.

The blootlist itself, is stored in nvram (non volatile ram), and contain entries to actual bootable devices.

Viewing the bootlist: (should be no problem);

```
# bootlist -m normal -o              # -m specifies which bootlist to display: for "normal" boot, or for "service" boot.

hdisk0
hdisk1
```

As the first item returned, you see here hdisk0 listed, which is a bootable harddisk.

If you need to check the bootlist in "service mode" (*a special bootoption, more on that later*), use:

```
# bootlist -m service -o
```

Changing the bootlist: (you have to be carefull: it cannot be done on all systems)

If you have a relatively "simple" configuration, using a local private disk for booting the system,
and there exists other local private disks (or devices), then changing the bootlist is probably simple.
That is not to say that it can't be done with virtualized scsi or "remote devices",
but we just can't let simple commands be representative for all cases. It just can't !

The bootlist, (in normal operations, at least for local private devices), can be changed using
the bootlist command, like for example:

```
# bootlist -m normal hdisk0 cd0
```

This command makes sure the hdisk0 is the first device used to boot the system, and secondly, from cd0.

```
# bootlist -m normal hdisk0 hdisk1
```

This command makes sure the hdisk0 is the first device used to boot the system, and could use hdisk1 as well.
It works this way: If no blv is detected on the first device, the system moves on
to the next device in the list.

If you want to change the bootlist for the system in service mode, you can change the list in order to use rmt0,
for example, if you need to restore the rootvg from a tape mounted in rmt0.

```
# bootlist -m service rmt0
```

Thus take notice of the keywords "normal" (thus normal boot) and "service" (boot to SMS).

For easy reference, here is the full syntax of the "bootlist" command:

```
bootlist [ { -m Mode } [ -r ] [  -o  ] [ [  -i ] [ -V ] [ -F ]| [ [ -f File ] [  Device [ Attr=Value ... ] ... ] ] ] [ -v ]
```

```
-m Specifies which boot list to display or alter. Like, as we have seen: "normal" and "service", but "both",
  'or "prevboot" exists as well.
-o Indicates that the specified boot list is to be displayed after any specified alteration is performed.
   The output is a list of device names.
```

9.3.2 The "bosboot" command:

The "bosboot" command usually is used to create a bootimage on a bootable device.
So, as a simplistic example, if hdisk0 must be bootable, or you want to be sure its bootable, use

```
# bosboot -ad /dev/hdisk0
```

-a Creates a complete boot image
-d let's you specify which device you want to make bootable.

You should also (if needed) update the list of bootable devices, using the "bootlist" command.
On a local disk, the boot logical volume is "hd5".

Here too, a warning can't hurt.
In a relatively simple environment with a couple of private local disks, there should not be problems.
In a complex environment, using SAN, virtualized scsi, or other types of storage, it may not be that simple as the above example suggests.

The bosboot command uses "/tmp" for processing, so if /tmp is too small, bosboot can fail.
ofcourse, that can be fixed easily.

In some rare cases, the new bootimage could be too large, to store in the presently sized BLV. This could happen on older disks with a small partition size.
An other reason may be that the ODM files have grown so large, that "savebase" cannot store copies on the bootdisks.
(See chapter 10)

9.4 Shutdown of an AIX LPAR:

As shown in fig 17, from the GUI on the HMC, if you *rightclick an LPAR*, you are able to "shutdown" a Partition (along with other options).

By far, the best way to orderly shutdown an AIX operating system, is doing that from a session in the OS, using appropriate commands.
If you do that, stop scripts will run and filesystems will be cleanly unmounted.

You can use the init, halt, reboot and shutdown commands to stop the AIX Operating System.
The "best" command to use is the shutdown command.

The "/etc/rc.shutdown" file:

If you need a customized shutdown sequence, you can create a file called /etc/rc.shutdown.
If this file exists, it is called by the shutdown command and is executed first.
This can be usefull for example, if you need to close a database prior to a shutdown.
If rc.shutdown fails (non zero return code value), the shutdown cycle is terminated.

Here is an example of a "rc.shutdown" file:

```
#cat /etc/rc.shutdown
```

```
#!/bin/ksh

# stop Control-SA/Agent
/etc/rc.ctsa stop
/etc/rc.mwa stop
/etc/rc.opc stop

# Stop TSM dsmcad en scheduler
/etc/rc.dsm stop

# Stop TSCM client
/opt/IBM/SCM/client/jacclient stop

# Stop App servers
/etc/rc.ihs stop
/etc/rc.ihs stop des
/etc/rc.appserver stop APPSRV1
/etc/rc.nodeagent stop
/etc/rc.dmgr stop

# Stop db2 instances
/etc/rc.db2_udb stop all

# Stop Oracle instance
export ORACLE_SID=sales
sqlplus /nolog <<EOF
connect / as sysdba
shutdown immediate;
exit 0
EOF

exit 0
```

As you can see, many specific rc scripts are called, or other commands are issued,
to stop different applications or services.

The shutdown command:

Per default the shutdown command will send a message (using 'wall" command) of the impeding shotdown to all users,
unless the "-F" flag is used. The default "wait time" before the shutdown actually starts, is 60 seconds.
The "-F" flag will omit the message and waittime (so it's a fast shutdown).
Here are a few examples using the shutdown command:

If you interactively will use the shutdown command, you might consider go to the "/" fileystem first,
so that you will not "hold" any filesystems.

To shutdown the system (no restart) and using the default of the message and one minute wait, use:

```
# shutdown
```

To shutdown the system (no restart) and using the default of the message and two minutes of wait, use:

```
# shutdown +2
```

Bring the system from multi-user mode to maintenance mode, use

```
# shutdown -m
```

To restart the system, use

```
# shutdown -r
```

To restart the system, where no message is send and there is no one minute waittime

```
# shutdown -Fr
```

When using shutdown, the system stops the accounting and error logging processes and writes an entry to the error log.
The shutdown command then runs the killall command to end any remaining processes and runs the sync command
to flush all memory resident disk blocks. Finally, it unmounts the file systems and calls the halt command.

Chapter 10. Some keypoints on Backup & Restore.

Here we can distinguish between "system related" backups (like "mksysb" which backups the rootvg), and "user related"
backups, where for example a user might create a backup of some filesystem (or directory),
using the "backup" or "tar" command.

The first "category" makes backups of system related items.
The second category, makes backups of user data or applications. Here we call them "the standard commands".

We don't *need* to make this "two way distinction": You may also say that there exists just a number of backup commands,
where each might be more appropriate to use in a certain situation.

10.1 System related backup and recovery:

Warning: Presented below, is only the "general" theory. No doubt in reality (on your systems),
it all will work "somewhat" different. If you haven't already done so, you should test
restores of mksysb backups on representative test lpars.

10.1.1 Creating a mksysb backup of the Operating System:

With "mksysb", you can create a bootable tape which contains all information to restore your AIX Operating System.
But you can use mksysb also to create a file, containing that data.
In other words: mksysb creates an installable image of the root volume group either in a file or onto a bootable tape.
If you backup to file, that file by itself is not bootable, but you could use it in a Network based install (NIM).

So, since it is a means to restore your complete OS (onto a LPAR, or system), AIX sysadmins will create a
mksysb backup either on a regular basis (e.g.: once a week), and before (and after) any fundamental change
occurs on the system, like installing a "Service Pack", or Technology Level update.

The mksysb command creates an installable image of the rootvg. This is synonym to say that mksysb creates
a backup of the operating system (that is, the root volume group).

The following is created:
The tape format includes a boot image (1), a bosinstall image (2), and an empty table of contents (3),

followed by the system backup (root volume group) image (4).
Below is a figure depicting those 4 "area's" or files.
Further more, mksysb will store an "image.data" file, and a "bosinst.data" file, on that tape.
- image.data: a file that describes the LV's and filesystems in rootvg
- bosinst.data: a control file for the bosinstall image.

You can use this backup to reinstall a system to its original state after it has been corrupted.
If you create the backup on tape, the tape is bootable *and includes the installation programs* needed to install from the backup.

Examples:

To generate a system backup and create an /image.data file (generated by the mkszfile command) to a tape device named /dev/rmt0, type:

```
# mksysb -i /dev/rmt0
```

If a backup tape was created with the -e switch, like in:

```
# mksysb -i -e /dev/rmt0
```

then a number of directories are NOT included in the backup. These exclusions are listed in the "/etc/exclude.rootvg" file.

The -i flag will call the "mkszfile" command, which produces the file "/image.data".
This file includes the sizes, names, maps, and mount points of logical volumes and file systems in the root volume group.
It will then be included in the mksysb backup.

To create a mksysb backup to a file, use a command similar to:

```
# mksysb -i /backups/system/n05lp06
```

This creates the file "n05lp06", which is a mksysb backup to a file. You could use it at a "NIM" based installtion.

The file-system image is in backup-file format, that is, it's the format as used with the AIX "backup" command.
The tape format includes a boot image, a bosinstall image,
and an empty table of contents followed by the system backup (root volume group) image. The root volume group image is in backup-file format, starting with the data files and then any optional map files.

Conceptual view of a mksysb backup:

There will be four images on the mksysb tape, and the fourth image will contain ONLY rootvg JFS or JFS2 mounted file systems. The target tape drive must be local to create a bootable tape.

The following is a conceptual view of mksysb's four images.

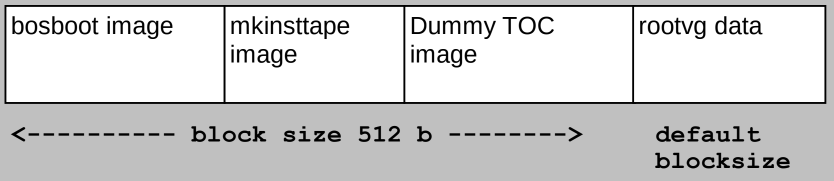


Fig. 22.

10.1.2 Restoring the full (or parts of a) mksysb backup:

Full Restore of a mksysb:

This is actually saying that you are going to restore rootvg from a mksysb backup.
One way to do that is the following.

If your system or lpar can use a tapedevice, either it currently owns it, or it can use the rmt of the HMC, proceed as follows:

-Insert the mksysb tape
-Power on the machine. The system will boot from the tape.
-The Installation and Maintenance Menu will be displayed.

```
                      Welcome to Base Operating System
                      Installation and Maintenance

Type the number of your choice and press Enter.  Choice is indicated by >>>.

    1 Start Install Now with Default Settings

    2 Change/Show Installation Settings and Install

>>> 3 Start Maintenance Mode for System Recovery
```

Type 3 and press enter to start maintenance mode.
The next screen you should see is :-

```
                    Maintenance

Type the number of your choice and press Enter.

>>> 1 Access a Root Volume Group
    2 Copy a System Dump to Removable Media
    3 Access Advanced Maintenance Functions
    4 Install from a System Backup

>>> Choice [1]:
```

Type 4 and press enter to install from a system backup.

Restore individual files from a mksysb backup:

As you can see from figure 22, the fourth file of the mksysb backup contains the rootvg data,
in the "backup" format (that is, the format that the backup command use).
You can also say that it's the third file, counting from "0".
You can restore files from a backup created using the "backup command", with the "restore command".

Suppose that you want to restore the file "/etc/filesystems" from the mksysb tape.

Proceed as follows:

```
# cd /                                                   # get to the root directory
# tctl -f /dev/rmt0 rewind                               # rewind the tape
# tctl -f /dev/rmt0.1 fsf 3                              # move the tape to the third file, no rewind (counting from 0)
# restore -xqf /dev/rmt0.1 -s 1 ./ect/filesystems        # extract the filesystems file
```

Further explanation why you must use the fsf 3 (fast forward skip file 3):
The format of the tape is as follows:
1. A BOS boot image
2. A BOS install image
3. A dummy Table Of Contents
4. The system backup of the rootvg

So if you just need to restore some files, first forward the tape pointer to position 3, counting from 0.

You can also restore a complete directory from the mksysb image. Use a command similar to:

```
restore -xvdf <mksysb.image> ./your/directory
```

The dot at the front of the path is important.
The "-d" flag indicates that this is a directory and everything in it should be restored. If you omit that, you'll restore an empty directory.

The directory will be restored underneath whatever directory you're in. So if you're in your home directory it might create:
/home/albert/your/directory.
So that's why you probably will go to the "/" root first.

Restore of a mksysb from a mirrored rootvg, to a new rootvg of 1 Physical Volume:

In reality, nummerous problems can popup (I don't want to be "dramatic" here, but in my experience, somehow it always does).
That's why you certainly must test mksysb restores on testmachines.

One common problem is this one:

The "image.data" file, consists of a description of all LV's, filesystems and mounts in the rootvg.
If your source rootvg was mirrored, but you need to use the mksysb on a rootvg consisting of only one PV, then you might use this trick (before any mishap happens).

So, if you ever suspect that you have to restore to a rootvg of only one disk, do this:

- create a 'image.dat' file in the root filesystem using:

```
# mkszfile
```

- Edit the "/image.data" file, and change the copies from "2" to "1".

- Create a mksysb tape, but without invoking mkszfile, so that the creation of "/image.data" is skipped;

```
# mksysb /dev/rmt0
```

This time, you have a mksysb tape, using your customized stanza file.

10.1.3 The image.data and bosinst.data files:

The image.data file:

The mkszfile command creates the "/image.data" file. Also, if you run mksysb using the -i flag,
it will call the "mkszfile" command.
This file includes the sizes, names, maps, and mount points of logical volumes and file systems in the root volume group.
It will then be included in the mksysb backup.

The bosinst.data file:

This file controls the AIX installation. It can also be used as an "answerfile" for a non-prompted installation.
Below is an example of a bosinst.data file:

```
control_flow:
    CONSOLE = Default
    INSTALL_METHOD = overwrite
    PROMPT = no
    EXISTING_SYSTEM_OVERWRITE = yes
    INSTALL_X_IF_ADAPTER = yes
    RUN_STARTUP = yes
    RM_INST_ROOTS = no
    ERROR_EXIT =
    CUSTOMIZATION_FILE =
    TCB = no
    INSTALL_TYPE =
    BUNDLES =
    RECOVER_DEVICES = no
    BOSINST_DEBUG = no
    ACCEPT_LICENSES = yes
    DESKTOP = NONE
    INSTALL_DEVICES_AND_UPDATES = yes
..
target_disk_data:
   PVID =
   CONNECTION =
   LOCATION =
   SIZE_MB =
   HDISKNAME = hdisk0
```

As we already know, running "mksysb -i" will invoke mkszfile to create the "/image.data" file.
The other data file that mksysb uses, is the "/bosinst.data" file. If a /bosinst.data file does not exist,
/var/adm/ras/bosinst.data is copied to "/". mksysb will always updates the target_disk_data stanzas in bosinst.data
to match the disks currently in the root volume group of the system where the mksysb command is running.

If you are using a customized /bosinst.data file and do not want the target_disk_data stanzas updated,
you must create the file "/save_bosinst.data_file".
The mksysb command then does not update /bosinst.data if the "/save_bosinst.data_file" exists.
Ofcourse, you can create that file using:

```
# cd /
# touch save_bosinst.data_file
```

10.1.4 make a copy of a Logical Volume (filesystem) using "cplv":

Please see section 1.3.

10.1.5 Backup and Restore of complete Volume Groups:

The "rootvg" volume group, should be saved using the mksysb command.

But on a machine, or lpar, you might have several Volume Groups.
If you want to backup a complete Volume Group, you can use the "savevg" command.
To restore such a backup, use the "restvg" command.

savevg:

The VG must be varied on, and the filesystems should be mounted.

The savevg and restvg commands usually use a datafile, a sort of listfile (that resembles the "image.data" file as is used with the mksysb command), that describes the VG.

Per default, that datafile is stored as "/tmp/vgdata/<vgname>/<vgname>.data".
So, suppose you have a VG named "applvg", this data file would be "/tmp/vgdata/applvg/applvg.data".
This file will be created by the "mkvgdata" command, which is called by "savevg" if the "-i" flag is used.

The full syntax is:

```
savevg [ -a ] [ -A ] [  -b Blocks ] [  -e ] [  -f Device ] [  -i |  -m ] [ -p ] [ -r ] [ -v ] [ -V ] [ -X ] VGName [-Z]
```

The savevg command can use two important switches (among many other switches):

-i Creates the data file by calling the mkvgdata command.
-f Specifies the device or file name on which the image is to be stored. The default is the /dev/rmt0 device.

So, if you use the "savevg" command with -i switch, the mkvgdata command is called,
which will create the datafile. Since that file should exist anyway, you probably will always use that switch.
And, with "-f" you can specify your backup destination, which can be a tape or a file.

Examples:

```
# savevg -if /dev/rmt0 applvg                  # backups applvg to tape.
# savevg -if /home/vgbackup datavg             # backups datavg to a file.
```

restvg:

You can restore a backup of a VG that was created with the 'savevg' command, by using the 'restvg' command.
As you have seen in the former section, the 'savevg' command creates a 'datafile' which also describes
on which physical disks (PV's) the Volume Group is located.
The restvg command will per default use the same datafile, and will restore on the same physical disk(s).
But you could override that with the "DiskName" parameter.

The full syntax is:

```
restvg [ -b Blocks ] [ -d FileName ][ -f Device ] [ -l ] [ -q ] [ -r ] [ -s ] [ -n ] [ -P PPsize ] [ DiskName ... ]
```

DiskName : Specifies the names of disk devices to be used instead of the disk devices listed in the vgname.data file.
-d FileName : this file will be used as the vgname.data file instead of the one contained within
'the backup image being restored.
-f Device : specifies the Device name of the backup media.
-l : displays useful information about a volume group backup.
-n : specifies that the existing MAP files are ignored.
-P PPsize : specifies the number of megabytes in each physical partition.
-q : specifies that the usual prompt not be displayed before the restoration of the volume group image.
-r : recreates a volume groups structure only.
-s : specifies that the logical volumes be created at the minimum size possible to accommodate the file systems.

Examples:

```
# restvg -f /dev/rmt1 hdisk2 hdisk3           # This command restores the volume group image from the /dev/rmt0 device,
                                              onto hdisk2 and hdisk3. So the DiskNames used, overrides
                                              the disks specified in the <vgdata.data> file.

# restvg -f /home/vgbackup                    # This command restores the volume group from the specified file, and uses
                                              the default datafile which describes the volume group, like which physical disks
                                              are used.
```

10.2 Some remarks about the standard backup & recovery commands.

Most commands in the section, are not specific to AIX. For example, the "tar" command can be found on any unix.
Contrary, a few commands are quite specific to AIX, like the "backup" and (the related) "restore" commands.

tar:

This is one of the old friends, of anyone working on unix.
You can create a good backup of objects (usually some whole directory with possible subdirectories), using the "tar" command.
It saves not only files and directories, but ownership and permissions (filemodes) as well.

It's most important flags are:

```
 -c create
 -r append
 -x extract
 -v verbose
 -t list
 -f specify device (tape, file on disk)
 -z if, while extracting, it was zipped as well, as in "file.tar.gz"
```

Examples:

```
# tar -cf /backups/john.tar /home/john        # creates a tar file "john.tar" from the whole contents in "/home/john"
# tar -cf /dev/rmt0 /home/john                # creates an archive on tape from the whole contents in "/home/john"

# tar -tvf /tmp/example.tar                   # list contents of example.tar to the screen

# tar -xvf /dev/rmt0 /home/john               # extract the archive on tape to /home/john
# tar -xvf /dev/rmt0 /tmp/appl.zip            # just restore one file from tape
# tar -xvf data.tar                           # extract, or untar, data.tar
```

Note: using the "-" is not neccesary.

cpio:

It uses one of three "modes":

```
copy-out    : cpio -o       to backup
copy-in     : cpio -i       to restore
passthrough : cpio -p       to pass files from one 'place' to another 'place'
```

Example of a "copy in":

Suppose you have the backup file "disk1.cpio" , and you need to restore (extract) it on a filesystem.

```
# cpio -idmv < disk1.cpio
```

Example of a "passthrough":

This is really a good application of the cpio command.
Some people use the "cp -R" command to copy a tree from one place to another place on the filesystems.
But, using cp, you can copy the files and dirs allright, but you loose ownership information.
Much better is using cpio, because it preserves all permissions and owners.

Suppose you have a directory (or filesystem) "/data/dira" and you want to copy it fully to "/backup/dira",
then use a command like:

```
# cd /data/dira
# find . | cpio -pdvm /backup/dira
```

backup and restore commands:

These commands are specific for AIX, although some other unixes have those as well.
I would say that these commands are the preferred way to "backup" and "restore" files and directories on AIX.
I think that this is especially true, if you backup a directory tree (on some filesystem) to another filesystem.
But ofcourse, you can backup to tape as well.
Ownership and filemodes (permissions) are preserved.
If you backup a directory tree to another filesystem (or directory), a backupfile will be created in that directory.
Note: The 3rd file (counting from 0) on mksysb tape (which is rootvg), is in the backup format.

```
There are a few important things to know here:

The -i flag:       - Without this flag, the command will backup by I-node.
                     There are some issues here, for example that the source filesytem should be unmounted.
                     Using the  -i flag means that you just want to backup by name (instead of I node) , which usually
                     is the best way to go, unless you want to create a full and incremental backup policy.
The -f flag:       - Specifies the device to backup to, like /dev/rmt0
The -u flag:       - You can create full and incremental backups as shown in later examples.
                     When the -u flag is used with the backup command, the system will do an incremental backup
                     according to the -level number specified. For example, a level 5 backup will only back up the
                     data that has changed after the level 4 was made.
                     Levels can range from 0 to 9. But the -u flag applies only to backups by i-node.
                     Using the -u flag, also will update the "/etc/dumpdates" file with information about the backup.
The -v flag:       - As usual, this means verbose output (on screen or to logfile)
The -q flag:       - Using this flag means to backup that the destination is ready. Otherwise you will
                     be prompted to press Enter.
```

So, creating a full backup by name (instead of I-node) means that you will always use the -ifvq flags.

The backup command can be used 'as is', but it is also often used with "find", where find produces
a list of all files (including in subdirs) and using "|" it will be piped to "backup" which will deal with them further.

Examples:

By I node:

```
# backup -0 -uf /dev/rmt0 /data      # On Sunday: backup by I node, incremental level 0
# backup -1 -uf /dev/rmt0 /data      # On Monday: backup by I node, incremental level 1
```

By name:

```
# cd /data
# find . | backup -ifvq /dev/rmt0

# cd /data
# find . | backup -iqf /backups/data/data_01012009.backup
```

Here is a restore example:

```
# restore -xdvqf /backups/savedata/data_010109.backup
```

pax:

The pax utility supports several archive formats, including tar and cpio.
The syntax for the pax command is as follows:

```
pax <mode> <options>
```

```
-r: Read mode. When -r is specified, pax extracts the filenames and directories found in the archive.
    The archive is read from disk or tape. If an extracted file is a directory, the hierarchy
    is extracted as well. The extracted files are created relative to the current directory.

-w: Write mode. If you want to create an archive, you use -w.
    Pax writes the contents of the file to the standard output in an archive format specified
    by the -x option.
```

-rw: Copy mode. When both -r and -w are specified, pax copies the specified files to
     the destination directory.

When neither -r or -w is specified, pax displays the filenames and directories
found in the archive file. The list is written to standard output.

```
-a = append to the end of an existing archive
-b = block size, multiple of 512 bytes
-c = you can specify filepatterns
-f = specifies the pathname of the input or output archive
-p <string> = aemo
```

a does not preserve file access time; e preserve everything: user id, group id, filemode bits; m does not preserve
  file modification times;
o preserve uid and gid

Examples:

To copy current directory contents to tape, use -w mode and -f

```
# pax -w -f /dev/rmt0
```

To list a verbose table of contents stored on tape rmt0, use None mode and f

```
# pax -v -f /dev/rmt0
```

To copy the olddir directory hierarchy to newdir, enter:

```
# pax -rw olddir newdir
```

dd:

The dd command is more likely for sysadmins to use, but the basics of this command is the following.
dd is used to copy just a specified number of bytes or blocks (possibly performing on-the-fly byte order conversions).
It is quite special also, because it can copy raw diskdevices as well.

In it's most simplistic form, the syntax is:

```
# dd if=<input file> of=<output file> <option=value>
```

Obviously "if" specifies the source, and "of" specifies the backup.
Some of the most important options to specify on the commandline are:

```
bs=blocksize       Specifies both the input and output block size.
skip=number        Skips the specified number of input blocks before starting to copy.
seek=number        Specifies the starting record to copy from after the specified skipped blocks.
count=number       Copies only the number of input blocks specified by that value.
```

Examples:

```
# dd count=1 bs=4k skip=31 seek=1 if=/dev/hd4 of=/dev/hd4        # copy one block to another block on hd4; actually copying
                                                                   the superblock in this example
# dd  if=/dev/rmt0  ibs=1024  obs=1024  of=/dev/rmt1             # copy the tape in rmt0 to a tape in rmt1
# dd if=/dev/raw_LogicalVolumeName of=/dev/rmt0                  # copy a raw LV to tape
```

Chapter 11. Some keypoints on logging and diagnosing.

11.1 The errdemon, errlog, and the "errpt" reporting tool:

This section deals on how to view the OS, hardware, and software errors on your system or LPAR.

The error reporting tool "errpt":

Your primary source of viewing event- and errorlogging, is using the "errpt" command.
This shows you errors and warnings (and informational records) related to the system (the OS and hardware).

If you use the "errpt" command without any flags or options, it produces a summary report
of all events. Here is an example:

```
# errpt           # Or use "errpt | more", because the list might be so long that's it better to pipe it to "more".
```

```
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
0EC00096   0130224007 P U SYSPFS         STORAGE SUBSYSTEM FAILURE
0EC00096   0130223507 P U SYSPFS         STORAGE SUBSYSTEM FAILURE
F7DDA124   0130223507 U H LVDD           PHYSICAL VOLUME DECLARED MISSING
52715FA5   0130223507 U H LVDD           FAILED TO WRITE VOLUME GROUP STATUS AREA
CAD234BE   0130223507 U H LVDD           QUORUM LOST, VOLUME GROUP CLOSING
613E5F38   0130223507 P H LVDD           I/O ERROR DETECTED BY LVM
BFE4C025   0130161207 P H sysplanar0     UNDETERMINED ERROR
0EC00096   0130161207 P U SYSPFS         STORAGE SUBSYSTEM FAILURE
BFE4C025   0130161107 P H sysplanar0     UNDETERMINED ERROR
FEC31570   0130161107 P H sisscsia2      UNDETERMINED ERROR
```

The "identifier" like "0EC00096", can be used to "zoom in" into a detailed report of such a record, like for example:

```
# errpt -aj BFE4C025 | more          # The output usually is long, so use pipe it to "more"
```

This produces a report on that particular error record.
If there are more records with the same identifier, the command will show the reports,
starting from the most recent, up to the oldest one.
here is a sample output:

```
LABEL:          SCAN_ERROR_CHRP
IDENTIFIER:     BFE4C025

Date/Time:       Sat Apr 25 20:06:22 ZOM 2009
Sequence Number: 1110
Machine Id:      00CDA84C4C00
Node Id:         starboss
Class:           H
Type:            PERM
Resource Name:   sysplanar0
Resource Class:  planar
Resource Type:   sysplanar_rspc
etc..
```

The errdemon:

Suppose you retrieve a listing of the processes, you will also see the 'errdemon':

```
# ps -ef
     UID   PID  PPID   C    STIME    TTY  TIME CMD
..
    root  4198     1   0   Oct 17      -  0:00 /usr/lib/errdemon
..
```

This process is indeed responsible for maintaining the errorlog.
If you want to see some details about that errorlog, use the following command:

```
# /usr/lib/errdemon -l
Error Log Attributes
--------------------------------------------
Log File                /var/adm/ras/errlog
Log Size                1048576 bytes
Memory Buffer Size      8192 bytes
```

So, the default errorlog is "/var/adm/ras/errlog".

To manual stop and start the daemon:

```
# /usr/lib/errstop
# /usr/lib/errstart
```

The errclear command:

You can truncate (remove all records) the errorlog with the following:

```
# errclear 0
```

To delete all entries in the error log classified as software errors, enter:

```
# errclear -d S 0
```

-d specifies the record types as H (hardware), S (software), O (errlogger messages), and U (undetermined).

To delete all entries in the error log classified as software errors an older than 30 days, enter:

```
# errclear -d S 30
```

The logfile can be manually cleared using the "errclear" command. However, ususally root has one ore
more crontab jobs which will clear the log in a certain way. Take a look at the following crontab entries:

```
0 11  *  *  * /usr/bin/errclear -d S,O 30
0 12  *  *  * /usr/bin/errclear -d H 90
```

The above example means that Software related entries, older than 30 days, are cleared.
Also, all other entries older than 90 days are cleared from the log.

errlogger command:

Sometimes you may want to place some text or comment in the errlog, for eaxmple, after an repair.
You can do that with the "errlogger" command, like in:

```
# errlogger The fibercard was replaced.
```

How the daemon works:

The ODM contains the configuration database for the errorlogging mechanism.
It is the file:
/etc/objrepos/SWservAt

The kernel (diagnostic extension, or device driver) writes error information to the special file "/dev/error".
The errdemon continuously keep watching this file.
Before an entry is written to the error log, the errdemon daemon compares the label sent by the kernel or application code to the information (templates) in the ODM Repository. If there is a match, the daemon collects additional data.
Before an entry is written to the log, the errdemon daemon retrieves the appropriate template from the repository, the resource name of the unit that detected the error, and detail data.
Then, an entry is written in the errlog.

11.2 The utmp, wtmp and failedlogin files:

The utmp file, the wtmp file, and the failedlogin file contain records with user and accounting information. When a user successfully logs in, the login program writes entries in two files.

- The /etc/utmp file, contains a record of users logged into the system.
  The "who" command uses the information from /etc/utmp file.

- The /var/adm/wtmp file, contains connect-time accounting records.
  The "last" command will use this file, to show you the historical login and logout info, since the last reboot.

- On an invalid login attempt, due to an incorrect login name or password, the login program makes an entry in the /etc/security/failedlogin file, which contains a record of unsuccessful login attempts.

Use the "last" command to get a historical view of logins/logouts, connecttime, and the used terminal.

```
# last | more
```

Use the who command to read the contents of the /etc/security/failedlogin file:

```
# who /etc/security/failedlogin
# who /etc/security/failedlogin > /tmp/failed_login.txt
```

- Shutdown:

The "acctwtmp" command also writes special entries in the /var/adm/wtmp file concerning
system shutdowns and startups.

11.3 The "alog" command, and various other (alog) logs:

bootlog and console log:

Messages generated during the AIX boot, are stored on disk. Also, during runtime, messages might be send to the console
You can use the alog command to view various logs,

```
# alog -L                      #  List the defined log types
# alog -o -t boot              #  View the boot log
# alog -o -t console           #  View the console log
```

You may also use:

```
# alog -f /var/adm/ras/bootlog -o | more
# alog -f /var/adm/ras/conslog -o | more
```

The logfiles themselves are thus:

```
"/var/adm/ras/bootlog"      The bootlog
"/var/adm/ras/conslog"      The console log
```

11.4 The "snap" command:

The "snap" command gathers system configuration information and compresses the information into a pax file.
The information gathered with the snap command may be required to identify and resolve system problems.
In principle, the snap file is supposed to be send to IBM, for troubleshooting purposes.

In normal conditions, the command "snap -gc" should be sufficient. The pax file will be stored in /tmp/ibmsupt

```
# snap -gc
```

will create the following file:

/tmp/ibmsupt/snap.pax.Z

You can then handle this file, like sending to IBM.
The "/tmp/ibmsupt" directory will actually contain a few subdirs, with individual files, and the "snap -gc" command
will comfortably pack those files into one compressed (Z) archive.

The -g flag gathers general system information, including the following:

- Error report
- Copy of the customized Object Data Manager (ODM) database
- Trace file
- User environment
- Amount of physical memory and paging space

- Device and attribute information
- Security user information

The -c flag will create a compressed pax image (snap.pax.Z file).

Using the -d flag, you can select another directory instead of "/tmp/ibmsupt".

If you see what is collected using -g, all parts can be done manually by yourself, like using "errpt" etc..

Using the "-a" flag will collect all information.

11.5 The "diag" command:

The diag command is specifically for hardware problem determination

Hardware diagnostics can be run in three different ways:

- The first way is concurrent mode, where the system is up and running with users online, all processes running, and all volume groups being used.

- The second way is service mode. This is when you have the machine with AIX running in maintenance mode, thus with the minimum amount of processes started and only rootvg varied on.

- The third way is stand-alone diagnostics from booting from CD-ROM, or a NIM image. This diagnostic method uses an other version of AIX, so any diagnostics run are totally independent of the AIX setup on the machine being tested.

Without an argument, diag runs as a menu-driven program.

```
Move cursor to selection, then press Enter.

Diagnostic Routines
  This selection will test the machine hardware. Wrap plugs and
  other advanced functions will not be used.
Advanced Diagnostics Routines
  This selection will test the machine hardware. Wrap plugs and
  other advanced functions will be used.
Task Selection(Diagnostics, Advanced Diagnostics, Service Aids, etc.)
  This selection will list the tasks supported by these procedures.
  Once a task is selected, a resource menu may be presented showing
  all resources supported by the task.
Resource Selection
  This selection will list the resources in the system that are supported
  by these procedures. Once a resource is selected, a task menu will
  be presented showing all tasks that can be run on the resource(s).

F1=Help F10=Exit F3=Previous Menu
```

The first option, will perform a basic check, and will not ask you to unplug anything.
The second option goes deeper, and you can be asked to unplug cables.
The third option leads to submenu's for many service procedures

With the fourth option, you can select a class of resources.

When choosing one of the Diagnostics options, you may furher choose between "Problem Determination" mode,
or "System Verification" mode.
The first option will check the system, in conjuction with errorlog entries.
The second option is typically used when a hardware module has been replaced.

The diag command can also used with flags. The syntax is:

```
diag [ -a ] | [ -s [ -c ] ] [ -E days ] [ -e ] | [ -d Device [ -c ] [ -v ] [ -e ] [ -A ] ] | [ -B [ -c ] ] |
[ -T taskname ] | [ -S testsuite ] | [ -c -d Device -L pending | complete ]
```

```
-A Advanced mode. Default is non-Advanced mode.
-a Processes the changes in the hardware configuration. For example, missing and/or new resources.
-B Tests the base system devices, such as planar, memory, processor.
-c Indicates that the machine will not be attended. No questions will be asked. Results are written
   to standard output. Normally used by shell scripts.
-d Device Names the resource that should be tested. the Device parameter is a resource name
   displayed by the lscfg command.
-E Days Number of Days used to search the error log.
-e Causes the device's Diagnostic Application to be run in Error Log Analysis mode.
-L pending | complete Log Repair Action for a resource specified with the -d and -c flags.
   Use pending if the part has been replaced, but it is not yet known if this part will remain in the system.
   Use complete if the part has been replaced and it is known to remain in the system.
-S testsuite Tests the Test Suite Group:
   1.Base System
   2.I/O Devices
   3.Async Devices
   4.Graphics Devices
   5.SCSI Devices
   6.Storage Devices
   7.Commo Devices
   8.Multimedia Devices
-S Causes the system to be tested in system Checkout mode.
-T taskstring Specifies A particular Task to execute. the taskstring depends on the particular Task to be executed.
-v system Verification mode. Default is Problem Determination mode.
```

Examples:

```
# diag -d hdisk2 -c

Starting diagnostics.
Ending diagnostics.
```

The absence of output means that no errors were found..

11.6 Viewing characteristics of processes, and resource usage:

If you want to "see" what a program is actually doing in terms of "syscalls" or "file opens" etc..,
we are more talking about "tracing" or "debugging" that program (or shell script).

That's not what we are talking about in this chapter. Here, we want to view "*characteristics*" of programs,
like how much memory it consumes, which libraries are in use, and that kind of stuff.

First of all, almost all AIX systems will have "topas" and "nmon" installed.
Both tools will show you (ascii) graphical information on cpu usage, disk usage, top consuming processes etc..
They are really great on viewing the status of your system in real time.

In AIX v. 5.3 or later, you might also use the "procmon" tool, which displays all processes with their
characteristics, and you can alter their state as well, like priority.

It's very easy to pinpoint the "top most" consuming processes (in terms of cpu, memory), using nmon, topas or procmon.
And you can view many other characteristics as well, using those utilities.

You know… this document is about "keypoints" only, so here are a few keypoints:

Quick view overall memory size / usage / memory related parameters:

```
# bootinfo -r
# lsattr -E -l mem0
# lsattr -E -l sys0 -a realmem
# lparstat -i
# svmon -G
# vmstat -v
# vmo -L
# prtconf
```

Quick view overall paging space usage:

```
# lsps -a
# lsps -s
# pstat -s
```

Quick view processor attributes:

```
# lparstat -i
# mpstat
# prtconf | grep proc
# pmcycles -m
# lsattr -El procx_(x is 0,2, etc..)
# lscfg | grep proc
# pstat -S
```

Characteristics of a process with process id "pid":

```
# proctree pid               Dipslays the children of process pid.
# procstack pid              Displays information of the stack.
# procmap pid                Displays memory usage of a process.
# procldd pid                Diplays all libraries loaded by this process.
# procflags pid              Displays a process tracing flags, and the pending and holding signals
# procsig pid                Lists the signal actions for a process.
# proccred pid               Displays the credentials under which this process runs.
# procfiles pid              Shows the opened files of this process.
# pfiles pid                 Show the open files that a process uses.
# procwdx  pid               Shows the current working directory of this process.

# ldd program_name           Displays which libraries a program needs.
```

Resource usage of processes using the "ps" command:

Everybody knows the ps command, like using as "ps -ef" to retrieve a long listing of all processes that are running on your system. The following commands might show some relevant resource usage:

```
# ps aux | more              Should show top consuming processes
```

Overall resource usage and Performance Tools:

curt:

The "CPU Utilization Reporting Tool" or "curt", needs a trace file as it's input. After analyzing this trace, it displays summary and detailed information on CPU usage.

The general procedure is like this:

1. Start the trace, like:

```
# trace <your_options> -o trace_file
```

2. Start the capture:

```
# trcon
```

3. Stop the capture:

```
# trcoff
```

4. Feed the trace to "curt"

```
# curt -i trace_file -o outputfile
```

procmon:

still to do

sar:

still to do

svmon:

still to do

11.7 Booting problems and LED codes:

Bootsequence:

As a schematic overview on the bootprocess of the OS, the following can be used as a guideline:

BIST
POST
FIRMWARE ->
-- Locate a bootdevice from bootlist in NVRAM and start initial boot

- When a valid boot device is found, the first record or program sector number (PSN) is checked.
If it is a valid boot record, it is read into memory and is added to the IPL control block in memory.
Included in the key boot record data are the starting location of the boot image on the boot device,
the length of the boot image, and instructions on where to load the boot image in memory.

- The boot image is read sequentially from the boot device into memory starting at the location
specified in NVRAM. The disk boot image consists of the kernel, a RAM file system, and base customized
device information (customized reduced ODM).

- Control is passed to the kernel, which begins system initialization.

- The kernel runs init, which runs phase 1 of the "/sbin/rc.boot" script.
When the kernel initialization phase is completed, base device configuration begins.

-- Base Device Configuration Phase:

The init process starts the rc.boot script. Phase 1 of the rc.boot script performs the base device configuration,
and it includes the following steps:

. The boot script calls the restbase program to build the customized Object Data Manager (ODM) database
  in the RAM file system from the compressed customized data.
. The boot script starts the configuration manager, which accesses phase 1 ODM configuration rules to configure
  the base devices.
. The configuration manager starts the sys, bus, disk, SCSI, and the Logical Volume Manager (LVM) and
  rootvg volume group configuration methods.
. The configuration methods load the device drivers, create special files, and update the customized data
  in the ODM database.

-- System Boot Phase:

The System Boot Phase involved the following steps:

The init process starts phase 2 running of the rc.boot script. Phase 2 of rc.boot includes the following steps:
.Call the ipl_varyon program to vary on the rootvg volume group.
.Mount the hard disk file systems onto their normal mount points.
.Run the swapon program to start paging.
.Copy the customized data from the ODM database in the RAM file system to the ODM database in the hard disk file system.
.Exit the rc.boot script.

- After phase 2 of rc.boot, the boot process switches from the RAM file system to the hard disk root file system.
- Then the init process runs the processes defined by records in the /etc/inittab file.

One of the instructions in the /etc/inittab file runs phase 3 of the rc.boot script,

```
  cat /etc/inittab | grep rc.boot
  brc::sysinit:/sbin/rc.boot 3 >/dev/console 2>&1 # Phase 3 of system boot
```

which includes the following steps:
.Mount the /tmp hard disk file system.
.Start the configuration manager phase 2 to configure all remaining devices.
.Use the savebase command to save the customized data to the boot logical volume
.Exit the rc.boot script.

At the end of this process, the system is up and ready for use.
If the "/etc/inittab" has a record referring to "rc.local", that script is executed.

Most important LED codes in the bootprocess:

For convenient reference, here are the Standard rootvg LV's / Filesystems:

LV name	mountpoint
hd4	/
hd5	BLV boot logical volume
hd6	paging
hd2	/usr
hd3	/tmp
hd1	/home
hd9var	/var
hd10opt	/opt
hd8	jsflog
hd11admin	/admin contains /admin/tmp on AIX 6.1

During the various fases at boottime, as described above, in case of an error, a LED code may be important:

Damaged boot image BLV	LED 201
Invalid bootlist	LED 223-229
reduced ODM from BLV copied into RAMFS:	LED 548
bootinfo -b is called to determine the last bootdevice	LED 511
ipl_varyon of rootvg	LED 551,552,554,556
mount /dev/hd4 on temporary mountpoint /mnt	LED 555,557
mount /usr, /var	LED 518
syncvg rootvg, or inittab problem	LED 553

```
tcp/ip is being configured, and there is some problem                         LED 581
Corrupted filesystem, corrupted JFS log                                       LED 551 555 557
Superblock corrupted, corrupted customized ODM database                       LED 552,554,556
```

Last phases in the boot is where cfgcon is called, to configure the console.
cfgcon LED codes include:
C31: Console not yet configured.
C32: Console is an LFT terminal
C33: Console is a TTY
C34: Console is a file on disk
C99: Could not detect a console device

```
If at boot:
- kernel initializes and takes control:                                       OK=299
- the restbase command is called to copy a partial image of ODM
  from the BLV into the RAMFS. If this operation is successful,
  the LED display shows 510; otherwise, LED code 548 is shown.                OK=510, ERROR=548
- bootinfo -b is called to determine the last bootdevice                      OK=511
- The rootvg volume group is varied on with the special version of the varyonvg
  command named the ipl_varyon command. If this command is successful,
  the system displays 517; otherwise, one of the following LED codes will
  appear: 552, 554, or 556, and the boot process is halted.                   OK=517, ERROR=551,552,554,556
- Root file system hd4 is checked using the fsck -f command. This will verify
  whether the file system was unmounted cleanly before the last shutdown. If
  this command fails, the system will display code 555.                       ERROR=555
- The root file system (/dev/hd4) is mounted on a temporary mount point (/mnt)
  in RAMFS. If this fails, 557 will appear in the LED display.                ERROR=557
- The /usr file system is verified using the fsck -f command and then
  mounted. If this operation fails, the LED 518 appears.                      ERROR=518
- The /var file system is verified using the fsck -f command and then
  mounted. The copycore command checks if a dump occurred. If it did, it is
  copied from default dump devices, /dev/hd6, to the default copy directory,
  /var/adm/ras. Afterwards, /var is unmounted.
- The primary paging space from rootvg, /dev/hd6, will be activated.
- The mergedev process is called and all /dev files from the RAM file system
  are copied onto disk.
- All customized ODM files from the RAM file system are copied to disk. Both
  ODM versions from hd4 and hd5 are now synchronized.
- Finally, the root file system from rootvg (disk) is mounted over the root file
  system from the RAMFS. The mount points for the rootvg file systems
  become available. Now, the /var and /usr file systems from the rootvg are
  mounted again on their ordinary mount points.
```

There is no console available at all these stages, so all boot messages will be copied to
alog. The alog command maintains and manages logs.

Chapter 12. User environment and security.

12.1 AIX v5/v6 Security and user environment related files:

For specific AIX v6 files and commands, see later sections.

/etc/security/environ	Contains the environment attributes for users.
/etc/security/lastlog	Contains the last login attributes for users.
/etc/security/limits	Contains process resource limits for users.
/etc/security/user	Contains extended attributes for users.
/usr/lib/security/mkuser.default	Contains the default attributes for new users.
/etc/passwd	Contains the basic attributes of users.
/etc/security/passwd	Contains password information.
/etc/security/login.cfg	Contains configuration information for login and user authentication.
/etc/utmp	Contains the record of users logged into the system.
/var/adm/wtmp	Contains connect time accounting records.
/etc/security/failedlogin	Records all failed login attempts.
/etc/motd	Contains the message to be displayed every time a user logs in to the system.
/etc/environment	Specifies the basic environment for all processes.
/etc/group	Contains the basic attributes of groups.
/etc/security/group	Contains the extended attributes of groups.
/etc/profile	Contains environment settings for all users
$HOME/.profile	Contains environment settings for a specific user.

"/etc/security/limits" and ulimit command:

The /etc/security/limits file is an ASCII file that contains stanzas that specify the
process resource limits for each user. These limits are set by individual attributes
within a stanza.

ulimit Command:

Purpose
Sets or reports user resource limits.

Syntax
```
ulimit [ -H ] [ -S ] [ -a ] [ -c ] [ -d ] [  -f ] [ -m ] [ -n ] [ -r ] [ -s ] [ -t ] [-u ][ Limit ]
```

Description
The ulimit command sets or reports user process resource limits,
as defined in the /etc/security/limits file. This file contains these default limits:

```
default:
  fsize = 2097151
  core = 2097151
  cpu = -1
  data = 262144
  rss = 65536
  stack = 65536
  nofiles = 2000
  threads = -1
  nproc = -1

john:
  fsize = 2097151
  core = 2097151
etc..
```

These values are used as default settings when a new user is added to the system.
The values are set with the mkuser command when the user is added to the system, or changed with the chuser command.

Note for troubleshooting purposes: if somehow, a process or user, cannot create a larger file, it is worth checking the "ulimit" for that account. For example, if a file should be created of 3GB size, but the process "mysteriously" stops at say 1.5 GB, it could be due to a ulimit restriction.
Check the ulimit restrictions with "ulimit -a", or use smitty.

"/etc/security/lastlog":

The /etc/security/lastlog file is an ASCII file that contains stanzas with the last login attributes for users. Each stanza is identified by a user name and contains attributes in the Attribute=Value form. Each attribute is ended by a new-line character, and each stanza is ended by an additional new-line character.

So, it contains attributes like tty, hostname, datetime etc.. Of the lastlogin of each user.

/etc/utmp, /var/adm/wtmp, /etc/security/failedlogin, and the last command:

The last command displays, in reverse chronological order, all previous logins and logoffs still recorded in the /var/adm/wtmp file. The /var/adm/wtmp file collects login and logout records as these events occur and holds them until the records are processed by the acctcon1 and acctcon2 commands as part of the daily reporting procedures.

The utmp file, the wtmp file, and the failedlogin file contain records with user and accounting information. When a user successfully logs in, the login program writes entries in two files.

- The /etc/utmp file, which contains a record of users logged into the system.
  The command who -a processes the /etc/utmp file.

- The /var/adm/wtmp file (if it exists), which contains connect-time accounting records.

- On an invalid login attempt, due to an incorrect login name or password, the login program makes an entry in the /etc/security/failedlogin file, which contains a record of unsuccessful login attempts.

Use the who command to read the contents of the /etc/security/failedlogin file:

```
# who /etc/security/failedlogin
# who /etc/security/failedlogin > /tmp/failed_login.txt
```

To clear the file use:

```
# cp /dev/null /etc/security/failedlogin
```

"/usr/lib/security/mkuser.default" and "/etc/security/user":

The /usr/lib/security/mkuser.default file contains the default attributes for new users. This file is an ASCII file that contains user stanzas. These stanzas have attribute default values for users created by the mkuser command. Each attribute has the Attribute=Value form.

```
# pg /usr/lib/security/mkuser.default
user:
pgrp = staff
groups = staff
shell = /usr/bin/ksh
home = /home/$USER

admin:
pgrp = system
groups = system
shell = /usr/bin/ksh
home = /home/$USER
```

The /etc/security/user file contains extended user attributes. This is an ASCII file that contains attribute stanzas for users. The mkuser command creates a stanza in this file for each new user and initializes its attributes with the default attributes defined in the /usr/lib/security/mkuser.default file.
Each stanza in the /etc/security/user file is identified by a user name, followed by a colon (:), and contains comma-separated attributes in the Attribute=Value form.
If an attribute is not defined for a user, either the default stanza or the default value for the attribute is used. You can have multiple default stanzas in the /etc/security/user file. A default stanza applies to all of the stanzas that follow, but does not apply to the stanzas preceding it.

The mkuser command creates an entry for each new user in the /etc/security/user file and initializes its attributes with the attributes defined in the /usr/lib/security/mkuser.default file. To change attribute values, use the chuser command. To display the attributes and their values, use the lsuser command.
To remove a user, use the rmuser command.

"/etc/security/environ" and "/etc/environment"

In "/etc/environment" you specify settings (environment variables) for all processes.

The /etc/security/environ file is an ASCII file that contains stanzas with the environment attributes for users. Each stanza is identified by a user name and contains attributes in the Attribute=Value form, with a comma separating the attributes.
Each attribute is ended by a new-line character, and each stanza is ended by an additional new-line character.

"/etc/profile" and "$HOME/.profile"

The /etc/profile file contains further environment variables, as well as any commands to run, that apply to all users. Use the /etc/profile file to control variables such as:

- Export variables
- File creation mask (umask)
- Terminal types
- Mail messages to indicate when new mail has arrived

Commands to be included in /etc/profile should be appropriate for all users of the system. An example of a command that you may want all users to run when they log in is the news command.

The $HOME/.profile file enables you to customize your individual working environment. The .profile file also overrides commands and variables set in the /etc/profile file. Use the .profile file to control personal settings such as:

- the prompt
- personalized umask etc..

12.2 mkuser and usrck:

Add a local user in AIX:

Ofcourse, smitty can be used to add a local user to the system, or use the "mkuser" command.

Syntax mkuser:

```
mkuser [ -R load_module ] [-a username] [ Attribute=Value ... ] Name
```

As we have seen in section 12.1, a number of files contains stanza's for new users, so actually the mkuser command is often quite "short". Unless you want to override a particular stanza attribute, by using "Attribute=Value".
As an example that most is already in the stanza, for example, you do not need to specify a home dir for the new user, because that is already in the file "/usr/lib/security/mkuser.default".

If the authentication actually occurs through an external entity, you must specify the "load_module" using -R.

The mkuser command does not create password information for a user. It initializes the password field with an * (asterisk). Later, this field is set with the passwd or pwdadm command.
New accounts are disabled until the passwd or pwdadm commands are used to add authentication information to the /etc/security/passwd file.

The /usr/lib/security/mkuser.default file contains the default attributes for new users. This file is an ASCII file that contains user stanzas. These stanzas have attribute default values for users created by the mkuser command. Each attribute has the Attribute=Value form. If an attribute has a value of $USER, the mkuser command substitutes the name of the user. The end of each attribute pair and stanza is marked by a new-line character.

There are two stanzas, user and admin, that can contain all defined attributes except the id and admin attributes.
The mkuser command generates a unique id attribute. The admin attribute depends on whether the -a flag is used with
the mkuser command.

A typical user stanza looks like the following:

```
user:
   pgroup = staff
   groups = staff
   shell = /usr/bin/ksh
   home = /home/$USER
   auth1 = SYSTEM
```

To create the john user account with the default values in the /usr/lib/security/mkuser.default file, type:

```
# mkuser john
```

To create the john account as an administrator, type:

```
# mkuser -a john
```

To create the john user account and set the su attribute to a value of false, type:

```
# mkuser su=false john
```

To create the john account that is identified and authenticated through the LDAP load module, type:

```
# mkuser -R LDAP john
```

The usrck command:

With this command you can verify the correctness of useraccounts.

Syntax:

```
usrck { -l [ -b ] |  -n | -p |  -t |  -y } { ALL | User ... }
```

The usrck command verifies the correctness of the user definitions in the user database files,
by checking the definitions for ALL the users, or for the users specified by the User parameter.
With the "-y" flag, you can fix errors.

The command first checks the entries in the /etc/passwd file. If you indicate that the system should fix errors,
duplicate user names are reported and disabled. Duplicate IDs are reported only, because there is no system fix.
The usrck command then verifies that each user name listed in the /etc/passwd file has a stanza in the /etc/security/user,
/etc/security/limits and /etc/security/passwd files. The usrck command also verifies that each group name
listed in the /etc/group file has a stanza in the /etc/security/group file. The usrck command using the -y flag creates
stanzas in the security files for the missing user and group names.

To verify that all the users exist in the user database, and have any errors reported (but not fixed), enter:

```
# usrck  -n ALL
```

To delete from the user definitions those users who are not in the user database files, and have any errors reported, enter:

```
# usrck  -y ALL
```

12.3 The AIX v 5.3 and AIX 6.1 "aixpert" command:

WARNING: Always test on a representative test LPAR, because the impact could be substantial.

This command is a security hardening tool, which incorporates many functions into one system.
It was available since v5.3 TL05, but many enhancements were made in v6, like LDAP integration, undo functionality etc..

The command uses XML files with security settings. The home of aixpert is "/etc/security/aixpert",
where in subdirectories those files are stored.

A very imporatnt file is "aixpertall.xml", stored in "/etc/security/aixpert/core/".
It contains the XML listings of all possible system settings.

To quickly illustrate the command: suppose you have the file "/etc/security/aixpert/core/mysec.xml",
containing security settings, then you can apply those settings with:

```
# aixpert -f /etc/security/aixpert/core/mysec.xml
```

But you can also apply a pre-defined model by using the "-l" flag and choosing for "high", or "medium" etc..

The syntax of the command is:

```
aixpert -l h|high | m|medium | l|low | d|default | s|sox-cobit [-n -o filename ] [ -a -o filename ] [ -p ]

aixpert -c [ -p ]

aixpert -u [ -p ]

aixpert -d

aixpert [-f filename ] [ -a -o filename ] [ -p ]
```

What's in the XML files?

The XML files contain settings or rules. For example, the rule about constraints on passwords (minimum length etc..),
along with any prerequiste filesets that are needed for implementing the rule:

```
<AIXPertEntry name="minagehls">
<AIXPertRuleType> 1 </<AIXPertRuleType>
<AIXPertRuleState>Desired</AIXPertRuleState>
<AIXPertDescription> Specifies the minimum number of weeks to 1 week,
before a password can be changed </AIXPertDescription>
<AIXPertPrereqList> bos.rte.date, bos.rte.com
etc..
```

Main Usage:

-l flag:
If you use `aixpert -l high|medium|low|default|sox-cobit`, then the config settings to that particular
security model (like "high" etc..) are applied to the system.

In combination with "-o" and "-n", you can write the records belonging to that specified security level, to an output file.

-f flag:
With this flag, you can specify a file with certain security settings, which are then applied to the system.

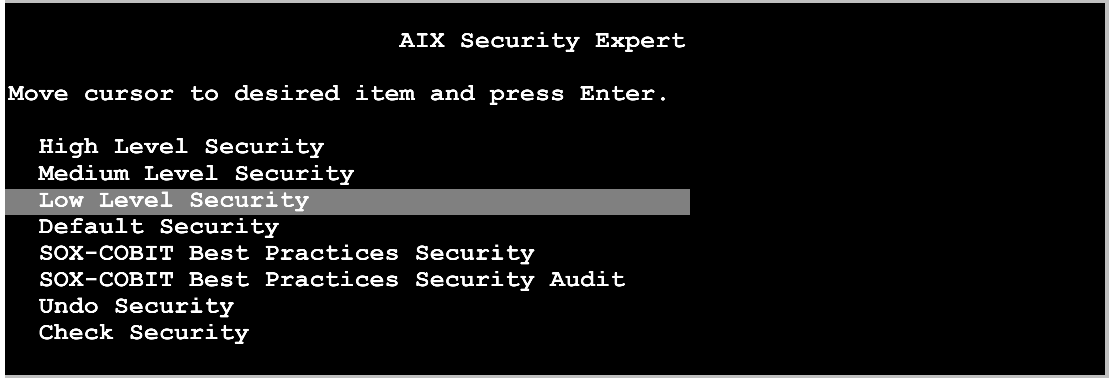
-o flag:
With this flag, you can specify a file to which you can export the security settings.

-u flag:
Undoes the security settings that have been applied.

Other interfaces to aixpert:

Instead of the command line, you can use smitty as well, using a guided menu system.

```
# smitty aixpert
```



```
                          AIX Security Expert

Move cursor to desired item and press Enter.

  High Level Security
  Medium Level Security
  Low Level Security
  Default Security
  SOX-COBIT Best Practices Security
  SOX-COBIT Best Practices Security Audit
  Undo Security
  Check Security
```

And you can also use the graphical environments of WebSM or IBM Systems Director Console.

Optionally LDAP integration:

An AIX Security Expert policy file can be created and saved in a central location on an LDAP server. The LDAP server stores the policy file containing the XML rules that is read by AIX Security Expert to determine security settings. Then as other systems in the network need to be hardened, the policy file is fetched from the LDAP server and a consistent policy is distributed and maintained throughout the network.

Examples:

Apply the settings from the file "mysec" to the system:
```
# aixpert -f /etc/security/aixpert/core/mysec.xml
```

Write the high security entries of your system to the file "MyPreferredSettings.xml".
```
# aixpert -l high -n -o /etc/security/aixpert/plugin/myPreferredSettings.xml
```

12.4 The regular Filemodes (file and directory permissions):

If, in some directory, you retrieve a filelisting using "ls -al", you see the familiar filemodes (or permissions) that are in effect on the files. Here I mean the "rwxrwxrwx" stuff

In short:

```
r = read    = 4
w = write   = 2
x = execute = 1
- = lack of that permission
```

A the mode of a file always "starts" with "-".
A the mode of a directory always starts with "d".

```
rwx   rwx   rwx
user  group others (world)
```

So, for edxample, if you see files with these mode:

```
-rwx-------       only the owner (user) can read, write, and execute the file (execute if applicable)
-rw-rw-rw-        the owner, group, and others can read an write the file.
-r--r-----        the owner, and the group, can only read the file
-rw-r-----        the owner can read and write, the group can only read
```

This type of access control is called "Discretionary Access Control" (DAC).
It can be further enhanced using "Access Control Lists" (ACL).

You can change the filemode (or permissions) using the chmod command. This one is really simple to use:
You just state what is added or subtracted to/from the user (owner) "u", the group "g", and others "o"

So, suppose "readme" has as filemode "-rw-------", and you want that other groupmembers may read that file as well, do:

```
# chmod g+r readme
```

And if the groupmembers may read and write that file, do"

```
# chmod g+rw readme
```

And if you want the groupmembers, and others, may read and write that file, do:

```
# chmod og+rw readme
```

And suppose that "myscript.ksh" presently has the permissions set as "-r-x------" (so only the owner may read and execute),
and you
allow the group to have read and execute as well, do:

```
# chmod g+rx myscript.ksh
```

So, in general the chmod command can be used as "chmod [u][g][o] + or - [r][w][x]"

+: adds a permission
-: revokes a permission

Since r=4, w=2, x=1, you can use the nummeric assignment as well. For example:

```
# chmod 644 myfile.txt
```

means that the permissions are set to "-rw-r--r--" because rw=4+2, r=4, r=4

Or what do you think of this:

```
# chmod 550 myscript.ksh
```

It will set the permission to "-r-xr-x---" because rx=4+1, rx=4+1, 0=nothing

Observe that in "nummeric" form, you use 3 digits. Now, actually, there is a "fourth" digit as well.
For this fourth digit, which determines the setuid or setgid modes, see the next section.

12.5 The setuid and setgid mode (or bits):

When setuid (set-user identification) permission is set on an executable file, a process that runs this file
is granted access based on the owner of the file, rather than the user who created the process.
This permission enables a user to access files and directories that are normally available only to the owner.

Note: on some unixes, the setuid bit feature is disabled per default.

The setuid permission is shown as an "s" instead of the "x" in the file permissions.

Example:

```
# ls -al

-r-sr-xr-x   1 johnny   accounting   12796 Jul 15 21:23 myprog
```

The setgid permission is similar to the setuid permission. The process's effective group ID (GID)
is changed to the group that owns the file, and a user is granted access based on the perms that are granted to that group.

Example:

```
# ls -al

-r-xr-sr-x   1 johnny   accounting   12796 Jul 15 21:23 myprog
```

You can use the chmod command again, to set the setuid or setgid bits.
Instead of a 3 digit mode, now use a 4 digit mode. <u>Use "4" for setuid, and "2" to set setgid.</u>

Example:

If I do this:

```
# chmod 555 myprog
# ls -al
-r-xr-xr-x   1 johnny   accounting   12796 Jul 15 21:23 myprog
```

So, this is the "normal" situation. But if I do this:

```
# chmod 4555 myprog
# ls -al
-r-sr-xr-x   1 johnny   accounting   12796 Jul 15 21:23 myprog
```

This time, the setuid is set, and the program will run under the "johnny" account at all times.

Note:
The following fameous example will explain it all. The "password" program has the setuid on.
The password program can indeed adjust the "/etc/security/password" file, which a ordinary
user is ofcourse not allowed to do. But if a user calls the password program, he/she can alter the password file.

```
# cd /usr/bin
# ls -al password
-r-sr-xr-x    1 root   security      40014 Apr 05 2009  /usr/bin/passwd
# cd /etc/security
# ls -al passwd
-rw-------    1 root   security       1293 Apr 11 11:22 /etc/security/passwd
```

You see? An ordinary user using passwd can change the passwd file even that it's only rw for root.

Is there something wrong with setuid? Most often, it's regarded as a security risk, because someone
can run a program with the credentials of the owner of the file.
Making shell scipts having setuid is often viewed as "very bad indeed".

The "fpm" program that is discussed in the next section, can be of help in hardening the system.
It will check the executables and daemons on the setuid and setgid bits, and change them according to the choosen policy.

12.6 The "fpm" program:

Essentially, the fpm, or "file permission manager" utility, can remove the setuid en setgid modes from programs and daemons.
Generally, this is described as "hardening" the system.

But beware that some programs *do need* the setuid, or depend on running commands which have the setuid bit in place.
Always test it on a representative test LPAR.

Fpm will check the executables and daemons on the setuid and setgid bits, and change them according to the choosen policy.

Syntax:

```
fpm [ -l high | medium | low | default [ -f file ] [ [ -c ] [ -p ] ] [ -v ] ] | [ -s ] | [ -q ] | [ -? ]
```

- Most stringent: -l high
This flag uses the list of files in the "/usr/lib/security/fpm/data/high_fpm_list" file
as input by default, but an alternate input file can be selected with the -f flag.

- Medium level: -l medium
This flag uses the list of files in the "/usr/lib/security/fpm/data/med_fpm_list" file
as input by default. An alternate input file can be selected with the -f flag.

- low level: -l low
This flag uses the list of files in the "/usr/lib/security/fpm/data/med_fpm_list" file
as input by default. An alternate input file can be selected with the -f flag.

- default: -l default
Returns the system commands previously modified by the fpm command to their default permissions,
if the commands were previously altered using the level of high, medium or low. This option reads the
/usr/lib/security/fpm/custom/default/*.* file and sets the permissions defined in the file.

-c flag: Checks the current situation against a certain specified level as "high", "medium" etc..

Examples:

```
# fpm -c -l low               # Checks the system against the low level specs, but makes no changes.

# fpm -l high                 # Applies the high level security settings on the system.
```

12.7 Extensions on DAC:

The filemode (-rwxrwxrwx stuff) determines whether the owner and/or group and/or others, may
read, write, or execute the file.
This falls in the realm of "Discretionary Access Control".

There are a few "enhancements" though:

1. Internal uid / gid checking:

There are a few special executables in AIX, that will not behave in the normal way of setting permissions.
What I mean is this.

If you look at a normal executable, and you change the permissions to 555 (or higher), then anyone
might execute that program.

For example, if I do this:

```
# chmod 555 myprog
```

Then the filemode is "-r-xr-xr-x" meaning that anyone can start myprog.

But there are a few exceptions, namely those programs that have uid / gid checking enabled, which
will make sure that ONLY a specific uid or gid, may execute the program.
For example, suppose a program will only execute if the uid=0, independent of how you have used the chmod command.
So, suppose "specialexec" is such an uid "aware" executable, and suppose it only "reacts" on a uid=0 that can activate it,
and we do this:

```
# chmod 555 specialexec
```

or we do this:

```
# chmod 777 specialexec
```

Then it won't work because 'specialexec" will check if you are uid (user id) 0. If not, it will not start.
The same "trick" may have been implemented using the group id (gid). When AIX is installed, a number of standard groups are present. In the same way, a program may check if the caller is member of a certain group.
If the caller is, then the program activates. If not, the program does not start, independent of the rwxrwxrwx permissions.

2. Access Control Lists (ACL):

The settings of regular filemode (-rwxrwxrwx), is not finegrained enough in certain situations.

Suppose the file "salaries.txt" is owned by john, who is member of the group accounting.
Suppose Ted is member of the group accounting.

Now, you want only Ted to have rw access to salaries.txt, but that access should not apply to the whole group accounting, and certainly not to all others.
How are you going to do that with the chmod command?

That's why ACL's are implemented in most unix versions. (Although in many cases, it's not even used)

With an ACL, you could create an "access list" for an object (like a file or directory) that looks like this:

```
base permissions:
  owner john: rwx
  group accounting: r--
  others: ---
extended permissions:
  ted: rw
```

So, you can view ACL's as an enhancement to regular filemodes (permissions).

You just use your EDITOR (like vi) to "edit" an acl. Actually, you need the EDITOR environment variable set.

Some important commands used to view, edit, or transfer acl's:

acledit	Modify the acl for an object
aclget	View the acl
aclput	Apply the existing acl of an object, to another object

Examples:

To list the ACL´s of the directory (filesystem) "/data/taxes":
```
# aclget /data/taxes
```

To modify an acl on an object, use a command similar to:
```
# acledit /data/taxes
```

To apply the same acl's of "/data1" to "/data2", use:
```
# aclget /data1 | aclput /data2".
```

Chapter 13. Some keypoints on printing.

In the AIX printing environment, files to be printed are sent to the AIX print
spooler daemon (qdaemon) using any of the AIX print commands (like enq, qprt, lp,
or lpr). The spooler daemon serializes the jobs. The spooler sends jobs, one at
a time, to back-end programs that may filter the data before sending it to the local
printer driver or network printing application.

You can check if the print spooler daemon is running with:

```
# ps -ef | grep qdaemon
..
    root 46980 43364   0   Oct 17      -  0:00 /usr/sbin/qdaemon
```

The daemon is started from /etc/inittab:

```
qdaemon:23456789:wait:/usr/bin/startsrc -sqdaemon
```

If you need to start the qdaemon, use the following command:

```
# startsrc -s qdaemon
```

Here are some important descriptions:

- print job:

A print job is a unit of work to be run on a printer.
The system assigns a unique job number to each job it runs.

- queue:

The queue is where you direct a print job. It is a stanza in the /etc/qconfig
file whose name is the name of the queue and points to the associated
queue device.

- queue Device

The queue device is the stanza in the /etc/qconfig file that normally follows
the local queue stanza. It specifies the /dev file (printer device) like "/dev/lp0" that should
be used.

- qdaemon

The qdaemon is a process that runs in the background and controls the
queues. It is generally started during IPL.

The main spooler command is the enq command. Although you can invoke
this command directly to queue a print job, three front-end commands are
defined for submitting a print job: The lp, lpr, and qprt commands. A print
request issued by one of these commands is first passed to the enq
command, which then places the information about the file in the queue for
the qdaemon to process.

The qdaemon uses backend processes (like "piobe") for internal handling of printing objects and printjobs.

- devices

In order to show the present devices, use the lsdev command:

```
# lsdev -Cc printer
lp0 Available 00-00-0P-00 Lexmark...
lp1 Available 00-00-S2-00 IBM...
lp2 Available 00-00-S1-00 Hewlett-Packard...
```

Individual device files can be listed with the ls command, for example

```
# ls -al /dev/lp0
crw-rw-rw- 1 root system 25,0 Oct 19 13:62 /dev/lp0
```

- The Print Configuration File

The file that holds the configuration for the printers that exist on the system is the /etc/qconfig file. It is the most important file in the spooler domain for these reasons:

- It contains the definition of every queue known to the spooler.
- A system administrator can read this file and discern the function of each queue.
- Although It is not recommended, this file can be edited to modify spooler queues without halting the spooler.

The /etc/qconfig file describes all of the queues defined in the AIX operating system. A queue is a named, ordered list of requests for a specific device. A device is something (either hardware or software) than can handle those requests one at a time. The queue provides serial access to the device.

The following is an example of the partial contents of the /etc/qconfig file.

```
..
..
lpforu:
device = lp0
lp0:
file = /dev/lp0
header = never
trailer = never
access = both
backend = /usr/lib/lpd/piobe
```

- Quick list of commands:

The following commands are usefull controlling printjobs and queue's:

Submit print jobs	Status print jobs	Cancel print jobs	Control printsubsystem
enq	enq -A	enq -x	qadm
qprt	qchk	qcan	stopsrc -s qdaemon
lp	lpstat	lprm	startsrc -s qdaemon
lpr	lpq	cancel	

- Creating a print queue:

Using smitty:

```
# smitty mkque
# smitty mkpq
```

Or using the commandline:

1. Add a queue using the mkque command. For example, the following command
will configure a remote queue. It configures just the queue and not the queue device:

```
# mkque -qlp -a 'host=starboss' -a 'rq=solar'
```

The -q flag specifies the name of the queue to be added (lp).
The -a flag specifies a line to be added to the queue stanza in the qconfig
file (host=starboss and rq=solar). These flags must be entered last when
entering the mkque command on the command line.

2. Add a queue device associated with the queue you have added, using the
mkquedev command. For the queue we added in the previous example, the
following command will add a device named lpdev that has /usr/lib/lpd/piobe
as its backend:

```
# mkquedev -qlp -dlpdev -a 'backend=usr/lib/lpd/piobe'
```

The -q flag specifies the name of the queue (this name must already exist)
to which the queue device is added. The mkquedev command automatically
adds the device=attribute to the specified queue stanza.

Examples of Commands:

lpstat:

```
System5: lpstat
BSD    : lpq
AIX    : qchk
```

But lpstat is also most often used in AIX.
The lpstat command displays information about the current status of the queue's and printers.

The lpstat command syntax is as follows:
lpstat [-aList] [-cList] [-d] [-oList] [-pList] [-r] [-s]
[-t] [-uList] [-vList] [-W]
An example of the lpstat command without any flags is as follows:

```
# lpstat
Queue Dev Status Job Files User PP% Blks Cp Rnk
------ ---- ------- --- ---------------- ------------ --- ---- -- ---
lpforu lp0 READY
```

qchk:

The qchk command displays the current status information regarding
specified print jobs, print queues, or users.

The qchk command syntax is as follows:
qchk [-A] [-L | -W] [-P Printer] [-# JobNumber] [-q] [-u UserName] [-w Delay]
An example of the qchk command to view the status of job # 442:

```
# qchk -#422
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
------- ----- --------- --- ------------------ ---------- ---- -- ----- --- ---
prtdumm null  READY
qstatus: (WARNING): 0781-350 Job 442 not found -- perhaps it's done?
```

lpq:

The lpq command reports the status of the specified job or all jobs associated with the specified UserName and JobNumber variables.
The lpq command syntax is as follows:
lpq [+ [Number]] [-l | -W] [-P Printer] [JobNumber] [UserName]
The following is an example of the lpq command without any flags.

```
# lpq
Queue Dev Status Job Files User PP% Blks Cp Rnk
------ ---- ------- --- ---------------- ------------ --- ---- -- ---
lpforu lp0 READY
```

To display a job number in the print queue lp0, enter:

```
# lpq  -P lp0

Queue     Dev     Status    Job     Files    User      PP    %      Blks    CP    Rnk
lp0       dlp0    running   39       motd     guest    10    83      12
```

Printing commands:

lp, lpr, qprt, enq

System5: lp
BSD   : lpr
AIX   : qprt

1. To submit a printjob, use either lp. lpr, or qprt. All jobs will go to the system default queue unless the PRINTER or LPDEST variables are set. You can also specify on the command line which queue ti ose.
Use -d with lp or use -P with qprt and lpr.
All the printcommands lp, lpr, and qprt, actually call the "enq" command, which places the print request in a queue.

To print multiple copies, use the "qprt -N #" or "lp -n #" command.
For lpr use just a dash followed by the number of copies, like "lpr - #".

Examples:

```
# qprt -P funjet /tmp/testfile
# lpr -P funjet /tmp/testfile
# lp -d funjet /tmp/testfile
```

enq:

This command has a very extended syntax. It is used in multiple ways, like controlling printing objects, and managing printjobs. You can also use it to move printjobs from one queue to another queue.

```
# enq -q -PHPQUE1
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
------- ----- --------- --- ------------------ ---------- ---- -- ----- --- ---
HPQUE1  hp@UT READY
```

Stopping and starting a printqueue using the "enq" command:

>>>> Stopping the Print Queue

In the following scenario, you have a job printing on a print queue, but you need to stop the queue so that you can put more paper in the printer.

```
# lpstat -vlpforu

Queue Dev Status Job Files User PP % Blks Cp Rnk
------ ---- -------- --- ---------------- -------- ---- -- ---- -- ---
lpforu lp0 RUNNING 3 /etc/passwd root 1 100 1 1 1
```

Disable the print queue using the enq command as shown in the following example.

```
# enq -D -P 'lpforu:lp0'
```

Checking the printer queue using the qchk command as shown in the following example.

```
# qchk -Plpforu
Queue Dev Status Job Files User PP % Blks Cp Rnk
------ ---- -------- --- ---------------- -------- ---- -- ---- -- ---
lpforu lp0 DOWN 3 /etc/passwd root 1 100 1 1 1
```

>>>> Starting the Print Queue

You have replaced the paper, and you now want to restart the print queue so that it will finish your print job. Here is how you would do this.

```
# lpstat -vlpforu
Queue Dev Status Job Files User PP % Blks Cp Rnk
------ ---- -------- --- ---------------- -------- ---- -- ---- -- ---
lpforu lp0 DOWN 3 /etc/passwd root 1 100 1 1 1

# enq -U -P 'lpforu:lp0'
# qchk -P lpforu
Queue Dev Status Job Files User PP % Blks Cp Rnk
------ ---- -------- --- --------------- -------- ---- -- ---- -- ---
lpforu lp0 RUNNING 3 /etc/passwd root 1 100 1 1
```

Other commands that can be used to stop and start a queue:

```
# smitty qstop
# smitty qstart
```

To stop a queue, you can use the qadm and disable commands as well. For example, to stop the queue PCL-mv200:

```
# qadm -D PCL-mv200
# disable PCL-mv200
```

qadm:

The qadm command is a front-end command to the enq command. This command brings printers,
queues, and the spooling system up or down and also cancels jobs. The qadm command
translates the requested flags into a format that can be run by the enq command.
So, again the "enq" command forms the basis.

```
qadm {  -G } | { [  -D Printer ] [  -K Printer ] [  -U Printer ] [  -X Printer ] }
```

-D: brings down the printer or queue
-U: brings up the printer or queue
-G: brings down the printing subsystem, when all current jobs are ready

To bring down the complete queueing system, use:
```
# qadm  -G
```

To stop a particular queue:
```
# qadm -D PCL-mv200
```

Cancel jobs:

```
System5: cancel
BSD    : lprm
AIX    : qcan
```

For example to cancel Job Number 127 on whatever queue the job is on, run

```
# qcan -x 127
# cancel 127
```

To cancel all jobs queued on printer lp0, enter

```
# qcan -X -Plp0
# cancel lp0
```

Moving a printjob from one queue to another queue:

You can use wsm, smitty, or the command line.

Using smitty:

```
# smitty qmov
```

Using the commandline:

```
qmov -m DestinationQueue -# JobNumber
qmov -m DestinationQueue -P Queue
qmov -m DestinationQueue -u User
```

For example:

To move job number 280 to print queue hp2, type:
```
# qmov -mhp2 -#280
```

To move all print jobs on print queue hpq1 to print queue hp22, type:
```
qmov -mhpq2 -Phpq1
```

Chapter 14. Some keypoints on networking.

Adapters and interfaces:

When an adapter is added to the system, a logical device is created in the ODM, for example
Ethernet adapters will show as follows:

```
# lsdev -Cc adapter | grep ent
ent0   Available 10-80   IBM PCI Ethernet Adapter (22100020)
ent1   Available 20-60   Gigabit Ethernet-SX PCI Adapter (14100401)
```

So you will have an adapter, and a corresponding interface, like for example
The Adapter is   : ent0
Then the interface is: en0

To list all interfaces on the system, use:

```
# lsdev -Cc if
en0 Defined   10-80  Standard Ethernet Network Interface
en1 Defined   20-60  Standard Ethernet Network Interface
et0 Defined   10-80  IEEE 802.3 Ethernet Network INterface
et1 Defined   20-60  IEEE 802.3 Ethernet Network INterface
lo0 Available        Loopback Network INterface
```

A corresponding network interface will allow tcpip to use the adapter.
Most of the time, we will deal with auto-detectable adapters, but in some cases an interface might
need to be created manually with

```
# smitty inet
# smitty mkinet
```

To change or view attributes like duplex settings, use

```
# smitty chgenet
```

```
                  Change / Show Characteristics of an Ethernet Adapter

Type or select values in entry fields
Press Enter AFTER making all desired changes.

  Ethernet Adapter                           ent0
  Description                                IBM PCI Ethernet Adapt
  Status                                     Available
  Location                                   10-80
  Hardware Transmit queue size               [64]
  Hardware Receive queue size                [32]
  Full duplex                                no
  Enable Alternate Ethernet address          no
  Alternate Ethernet address                 [0x000000000]
  Apply change to DATABASE only              no
```

To show IP parameters of an interface, use:

```
# ifconfig -a                # shows all
# ifconfig en0               # shows specific interface
en0:
flags=e080863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT>
inet 195.116.119.2 netmask 0xffffff00 broadcast 195.116.119.255
```

If you want to view statistics and media attributes of an interface, you can use the "entstat" command, like:

```
# entstat -d en0
```

Resolving hostnames:

>> the "/etc/netsvc.conf" file and the NSORDER variable:

The default order in resolving host names is:

- BIND/DNS (named)
- Network Information Service (NIS)
- Local /etc/hosts file

The default order can be overwritten by creating the configuration file, /etc/netsvc.conf and specifying the desired order. Both the default and /etc/netsvc.conf can be overwritten with the environment variable NSORDER.

You can override the order by creating the /etc/netsvc.conf file with an entry.
If /etc/netsvc.conf does not exist, it will be just like you have the following entry:

```
hosts = bind,nis,local
```

You can override the order by changing the NSORDER environment variable. If it is not set,
it will be just like you have issued the command:

```
export NSORDER=bind,nis,local
```

>> the /etc/resolv.conf file:

If you use name services, you can provide the minimal information needed through the mktcpip command.
Typically, the "/etc/resolv.conf" file stores your domain name and name server ip addresses.
The mktcpip command creates or updates the /etc/resolv.conf file for you.

Starting and stopping NFS:

The following subsystems are part of the nfs group: nfsd, biod, rpc.lockd, rpc.statd, and rpc.mountd.
The nfs subsystem (group) is under control of the "resource controller", so starting and stopping nfs
is actually easy:

```
# startsrc -g nfs
# stopsrc -g nfs
```

inetd daemon:

After you have edited "/etc/inetd.conf", for example, to enable or disable some service,
you need to restart, or refresh inetd, to read the new configuration information.
To let inetd to reread the configfile:

```
# refresh -s inetd
```

If you need to stop and start inetd, use:

```
# stopsrc -s inetd
# startsrc -s inetd
```

Example of the "mktcpip" command:

The mktcpip command is a quick way to configure an interface.

```
# mktcpip -h starboss -a 10.10.10.5 -m 255.255.255.0 -i en0 \
-n 10.10.10.254 -d abc.xyz.nl -g 10.10.10.254 -s -C -A no
```

Activating, Deactivating, and Detach a network interface:

It might happen that you want to deactivate an interface. You can then activate it at a later time.
Here, you can use the "ifconfig" command, like:

```
# ifconfig en0 down
# ifconfig en0 up
```

It's also possible to specify network parameters, although smitty is used more often.

```
# ifconfig en0 10.10.10.30 netmask 255.255.255.0 up
```

When you do not need an interface anymore, you can "detach" it. It will then no longer be shown in an interface listing, like for example:

```
# ifconfig tr0 detach
```

The "/etc/hosts.equiv" and the "$HOME/.rhosts" files:

This is a bit old stuff, because in reality it is hardly used anymore because of security risks, but it might be an exam objective.

The /etc/hosts.equiv file, along with any local $HOME/.rhosts files, defines the hosts (computers on a network) and user accounts that can invoke remote commands on a local host without supplying a password. A user or host that is not required to supply a password is considered trusted.

When a local host receives a remote command request, the appropriate local daemon first checks the /etc/hosts.equiv file to determine if the request originates with a trusted user or host. For example, if the local host receives a remote login request, the rlogind daemon checks for the existence of a hosts.equiv file on the local host. If the file exists, but does not define the host or user, the system checks the appropriate $HOME/.rhosts file. This file is similar to the /etc/hosts.equiv file, except that it is maintained for individual users.
Both files, /etc/hosts.equiv and $HOME/.rhosts, must have permissions denying write access to group and other (600). If either group or other have write access to a file, that file will be ignored.

Examples:

In the "/etc/hosts.equiv" file, the following entries could be present:

For example, to allow all the users on the hosts starboss and cygnus to log in to the local host, you would enter:

```
starboss
cygnus
```

But if you only want to allow the user bob to log in from the host cygnus, you would enter:

```
starboss
cygnus bob
```

If you want to additionally also allow the user john to log in from any host, you would enter:

```
starboss
cygnus bob
+ john
```

Initialization of TCPIP at boot:

At IPL time, the init process will run the /etc/rc.tcpip after starting the SRC.
This is so because in the "/etc/inittab" file, the following record is present:

```
rctcpip:23456789:wait:/etc/rc.tcpip > /dev/console 2>&1 # Start TCP/IP daemons
```

The /etc/rc.tcpip file is a shell script that uses SRC commands to initialize selected deamons.
It can also be executed at any time from the command line.
These deamons are:

inetd (started by default),gated,routed,named,timed,rwhod

There are also deamons specific to the bos or to other applications that can be started through
the rc.tcpip file. These deamons are lpd, portmap, sendmail, syslogd (started by default)

The subsystems started from rc.tcpip can be stopped and restarted using the stopsrc and startsrc commands.

The telinit command:

If you want to change the "runlevel" of your machine, you can use the "telinit" command to do so.
For example, maybe you want to run the machine in "single user" or "maintenance mode".
Actually, the "telinit" is linked to the familiar "init", which is used on most other platforms
to reboot or change the runlevel.

```
{ telinit | init } { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | h | Q | q | S | s | M | m | N }
```

```
S,s,M,m: maintenance mode
0,1     : reserved
2       : normal
3-9     : user defined
Q,q     : reparse inittab file
```

AIX is a little bit different from most other unixes, in the sense that here level 2 is the default multi-user level.

The nslookup command:

The nslookup command queries domain name servers for information about
various hosts and domains. The nslookup command is useful for determining
host names of servers by IP address, host name, or
domain. The nslookup command can be run as follows:

```
# nslookup [IPAddress | HostName]
```

For example, to determine the host name of the system with the IP address of
202.25.200.128, you would enter:

```
# nslookup 202.25.200.128
Server: ns01.antapex.org
Address: 9.3.240.2
Name: lp05.antapex.org
Address: 202.25.200.128
```

The no command:

The no command is used to configure network attributes. The no commands sets or displays current
network attributes in the kernel. It will only operate on the currently running kernel.
Whether the commands sets or displays an attribute is determined by the accompanying flag:
the -o flag performs both actions.

The syntax is much like the syntax used with the vmo command (see chapter 3).

Some examples of setting tunables:

```
# no -o thewall=3072
# no -o tcp_sendspace=16384
# no -o ipqmaxlen=512
# no -o ipforwarding=1               # Meaning IP forwarding is true.
```

Most important flags:

```
-a                            Displays parameters.
-L [tunable]                  Displays all tunables, or the one specified with [tunable]
-o Tunable[=new_value]        Displays the value or sets the Tunable to new_value.
```

Adding network routes:

To add static network routes in order for your system to reach remote networks via a router / gateway, you can use smitty, or the route add command.

```
# smitty route
# smitty mkroute

# route add -net 192.168.1 -netmask 255.255.255.0 10.10.10.1
```

Well, that's it. Although this document is rather limited in scope,
I hope it was a bit usefull.