43

AIX update

May 1999

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 550955
From USA: 01144 1635 33823
E-mail: HarryLewis@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about AIX, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you actively be helping the free exchange of information, which benefits all AIX users, but you will also gain professional recognition for your expertise and that of your colleagues, as well as being paid a publication fee – Xephon pays at the rate of £170 ($250) per 1000 words for original material published in AIX Update.

To find out more about contributing an article, see *Notes for contributors* on Xephon's Web site, where you can download *Notes for contributors* in either text form or as an Adobe Acrobat file.

Editor

Harold Lewis

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1995 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

AIX Update on-line

Code from *AIX Update* is available from Xephon's Web page at www.xephon.com (you'll need the user-id shown on your address label to access it).

AIX’s ‘alt_disk_install’ feature

The **alt_disk_install** feature was introduced in AIX 4.3 in October 1997. This useful utility has been revised in each subsequent release of AIX. The most recent release, 4.3.2.0, is distributed on the AIX 4.3.2 Update CD-ROM in the *bos.alt_disk_install.rte*, *bos.msg.en_US.alt_disk_install.rte*, and *bos.alt_disk_install.boot_images* filesets.

ALT_DISK_INSTALL FUNCTIONALITY

The **alt_disk_install** command is used to install additional copies of the operating system to a dedicated disk or disks without interrupting the running version of the operating system. Two modes of operation are supported: ‘mksysb image restore’ and ‘root volume group cloning’.

The **mksysb**-based alternative disk installation restores the back-up image tape created using the **mksysb** command on another system to a disk or disks that are not currently in use. After the restore it is possible to boot the system from the restored disks.

The cloning option allows the user to create a back-up copy of the root volume system of the running machine. Once the copy is made, it is possible to install updates or additional application software on the alternative disk or disks.

ALTERNATIVE MKSYSB DISK INSTALLATION

The alternative **mksysb** mode of installation installs a **mksysb** image that was created on a different computer on the alternative disk or disks of the target machine. The alternative disk or disks cannot participate in a volume group that’s already defined. Note that the source system must have the same hardware configuration as the target system – if this is not the case, then all device and kernel support should be installed explicitly on the target system.

The **alt_disk_install** command performs following actions:

1 First it restores the *image.data* file from the **mksysb** image, unless a custom *image.data* is specified as one of the command’s arguments.

2 It then defines the *altinst_rootvg* volume group, which contains the disks to which the restore is performed.

3 The command then defines the logical volumes and filesystems that appear in *image.data*. In order to create unique logical volume and filesystem names, the prefix *alt_* is added to logical volume names and */alt_inst* to filesystem names. For example, *hd5* is created as *alt_hd5* and the file system */usr* as */alt_inst/usr*. The original names are restored at the end of the alternative disk installation process.

4 Next, files are restored from the **mksysb** installation image to populate the alternative filesystems. The boot image is copied to the boot logical volume of the newly created volume group, and the boot record of the boot disk is modified to allow booting from the disk. At this point, it is possible to run a customization script specified by the user. The alternative filesystems are mounted using names that conform to the standard */alt_inst/real_ filsystem_name*. Note that, at this time, it is impossible to install OS updates and applications as the kernels and libraries that are running do not conform to the level that is needed to support the alternative install image.

5 The filesystems are now unmounted, and the original names of logical volumes and filesystems are restored. The logical names are exported from the Object Data Manager (ODM), though the *altinst_rootvg* is only varied off. The boot list is then reset to point to the alternate root volume group boot disk. This is the default, which can be changed by the user. If requested, the system reboots at this point. When the system is rebooted, the **bosboot** command is run early in the boot process, and the system reboots again. This is done in order to synchronize the boot image with the **mksysb** that was restored.

6 After rebooting, the old *rootvg* volume group is renamed *old_rootvg* and *altinst_rootvg* is renamed *rootvg*.

ALTERNATIVE DISK ROOTVG CLONING

The procedure to clone *rootvg* to an alternative disk differs from that for restoring a **mksysb** image in three important respects:

1 An *exclude.list* file may be provided as argument to the **alt_disk_install** command. This specifies the files that are to be excluded from the cloning process.

2 It is possible to invoke the **installp** command after cloning *rootvg* to install updates, fixes, and new filesets in the alternative root volume group.

3 There is no need to run the **bosboot** command during the first boot after cloning.

PHASED ALTERNATIVE DISK INSTALLATION

Alternative disk installation comprises three phases. The default is to perform all three phases in one invocation of the command. The phases are:

Phase 1	Create the *altinst_rootvg* volume group, the *alt_* logical volumes, and the */alt_inst* filesystems. Restore a **mksysb** image or back up and restore *rootvg* data.
Phase 2	Run a customization script, if required. If cloning *rootvg*, install new filesets, fixes, and updates. Copy the *resolv.conf* file and all files necessary to remain a NIM client, if requested by the user.
Phase 3	Unmount the */alt_inst* filesystems, rename file systems and logical volumes, remove the *alt_* logical volume names from the ODM, and 'vary off' the *altinst_rootvg* volume group. Set the boot list and reboot (if specified).

The phases can be executed in following ways:

- Each phase may be executed separately.
- Phases 1 and 2 may be executed in one operation.
- Phases 2 and 3 may be executed in one operation (Phase 2 may be executed more than once before phase 3 is executed).
- All phases may be executed in one operation.

After execution phases 1 and 2, the */alt_inst* filesystems remain mounted. Phase 3 must be executed to obtain usable *rootvg*.

FLAGS AND EXAMPLES

Below are explanations of **alt_disk_install** command options and the tasks it can perform.

CREATING AN ALTERNATIVE DISK FROM A MKSYSB IMAGE

The command for this is as follows (note the use of the continuation character, '➤', to indicate that one line of code maps to several lines of print):

```
alt_disk_install  -d <device> [-i <image.data>] [-s <script>]
➤  [-R <resolv_conf>] [-D] [-B] [-V] [-r] [-p <platform>]
➤  [-L <mksysb_level>] [-n] [-P <phase_option>] <target_disk(s)>
```

Flags

- *–d <device>*
  Specifies the device from which to restore **mksysb**; this can be either a tape drive or the path to a **mksysb** image in the filesystem.
- *–I <image_data>*
  Specifies the full path to the *image.data* file that's to be used instead of the default one.
- *–s <script>*
  Specifies the full path to the customization script that needs to be run after **mksysb** is restored. The script can access file systems mounted under */alt_inst* before the system reboots.
- *–R <resolv.conf>*
  Specifies the full path to the *resolv.conf* file that's to be copied to */alt_inst* once **mksysb** is restored.
- *–D*
  Turns on debugging output.
- *–B*
  Directs the script not to set the boot list to the newly created alternative volume group. This flag cannot be used in the same

command as **-r**.

- *–V*
  Turns on verbose output. This flag directs the script to display files being backed up and restored.

- *–r*
  Directs the script to perform a reboot once the command has been executed.

- *–p <platform>*
  Specifies the platform type, which is necessary when creating the boot image name of the alternative disk. The default value can be obtained by executing the command **bootinfo -T** on AIX 4. 1 and **bootinfo -p** on AIX 4.2 and higher.

- *–L <mksysb_level>*
  Specifies the 'level', which is combined with the platform type to create the boot image name. Must be in the form *V.R.M*. The **mksysb** image is checked again at this level to verify that it's correct.

- *–n*
  Specifies that the system is a NIM client and is to remain so after reboot. The files */.rhosts* and */etc/niminfo* are copied to the alternate *rootvg*'s filesystem.

- *–P <phase>*
  Specifies the phase to execute in this invocation of **alt_disk_install**. Valid values are: *1*, *2*, *3*, *12*, *23*, and *all*.

Parameters

- *<target_disks>*
  Specifies the names of the disks that are to comprise the alternative root volume group. These disks must not be part of an existing volume group, and so must be shown in output of the **lspv** command as belonging to volume group *None*.

Example

Use the command below to install an alternative volume group from a **mksysb** tape (device */dev/rmt0*) to *hdisk1* and then run the customization script */tmp/custscript*.

```
at_disk_install -d /dev/rmt0 -s /tmp/custscript hdisk1
```

CREATING AN ALTERNATIVE DISK BY CLONING

```
alt_disk_install -C [-i <image.data>] [-s <script>]
➤  [-R <resolv_conf>] [-D] [-B] [-V] [-r] [-b <bundle_name>]
➤  [-I <installp_flags>] [-l images_location] [-f <fixes_bundle>]
➤  [-F<fixes>] [-e <exclude_file>] [-w <filesets>] [-n]
➤  [-P <phase_option>]  <target_disk(s)>
```

Flags

Note that only flags specific to cloning are listed below.

- *–b <bundle_name>*
  Specifies the full path to the optional file that contains the list of filesets of packages to be installed once cloning is complete. The **-l** flag must be used along with this flag.

- *–e <exclude_list>*
  Specifies the full path to the optional *exclude.file* that contains patterns for the names of files to be excluded from cloning.

- *–f <fix_bundle>*
  Specifies the full path to the optional file that contains a list of APARs to be installed once cloning is complete. The **-l** flag must be used along with this flag.

- *–F <fixes>*
  Specifies the optional list of APARs to be installed once cloning is complete. The **-l** flag must be used along with this flag.

- *–I <installp_flags>*
  Specifies the flags to use with the **installp** command when installing files and updates once cloning is complete. The default value is **-acgX**. The **-l** flag must be used along with this flag.

- *–l <images_location>*
  Specifies the location of **installp** images or updates that are to be applied after the clone operation. This can be the full path to a directory or a device name.

- *–w <filesets>*
  Specifies an optional list of filesets to be installed once cloning is complete. The **-l** flag must be used along with this flag.

Example

To clone the alternative volume group to *hdisk2* and then install the fixes listed in the */tmp/fixes* (located in */usr/sys/inst.images*), use the command:

```
at_disk_install -C -f /tmp/fixes -l /usr/sys/inst.images hdisk2
```

DETERMINE BOOT DISK OF THE VOLUME GROUP

```
alt_disk_install -q <disk>
```

You can use the **-q** flag to discover the boot disk of the volume group *old_rootvg* after rebooting from the alternative disk. For instance, if the output of the **lspv** command is:

```
hdisk0     00008061bcd8a785   old_rootvg
hdisk1     000764abcd345678   rootvg
hdisk2     000085b7a645ec28   old_rootvg
```

and the output of the **alt_disk_install –q hdisk0** command is *hdisk1*, then the boot disk of *old_rootvg* is *hdisk1*. Therefore, if you need to reboot from the original *rootvg*, use the command sequence below.

```
bootlist -m normal hdisk1
reboot now
```

RENAMING AN ALTERNATIVE DISK VOLUME GROUP

```
alt_disk_install -v <new_volume_group_name> <disk>
```

Once the alternative disk volume group is created, its name is set to *altinst_rootvg*. If you need to maintain a number of different alternative disks on one system (perhaps one installed with AIX 4.2.1, another with AIX 4.3, etc), then you need to rename alternative disks using meaningful names.

For instance, if the initial output of the **lspv** command is:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   None
```

```
hdisk2     000085b7a645ec28   None
hdisk3     0007abc346efadd7   None
```

after installing the alternative disk from **mksysb** using the command:

```
alt_disk_install -d /dev/rmt0 hdisk1
```

the output of **lspv** becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   altinst_rootvg
hdisk2     000085b7a645ec28   None
hdisk3     0007abc346efadd7   None
```

If the following command is now executed:

```
alt_disk_install -v alt_disk_421 hdisk1
```

then the output of **lspv** becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   alt_disk_421
hdisk2     000085b7a645ec28   None
hdisk3     0007abc346efadd7   None
```

If the user now clones *rootvg* using the command:

```
alt_disk_install -c hdisk2
```

the output of **lspv** now becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   alt_disk_421
hdisk2     000085b7a645ec28   altinst_rootvg
hdisk3     0007abc346efadd7   None
```

Now, after executing the following command to rename the alternative disk volume group:

```
alt_disk_install -v alt_disk_432 hdisk2
```

the output of **lspv** becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   alt_disk_421
hdisk2     000085b7a645ec28   alt_disk_432
hdisk3     0007abc346efadd7   None
```

CLEANING UP ALTERNATIVE DISK VOLUME GROUP

```
alt_disk_install -X <volume_group>
```

The **-X** flag of the **alt_disk_install** command provides the only supported method of removing an alternative volume group safely. When the command is invoked with this flag, all information about the alternative volume group is removed from the ODM. Note that the data itself is not removed from the affected disks, which means that the user can still reboot the system from that volume group, providing he or she changes the system's boot list. The example below illustrates the effect of **-X** flag.

If output of the **lspv** command is:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   old_rootvg
```

then, after executing the command:

```
alt_disk_install -x old_rootvg
```

the output of **lspv** becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   None
```

'WAKING UP' A VOLUME GROUP

```
alt_disk_install  -W  <disk>
```

The **-W** flag lets the user 'wake up' the alternative volume group to allow data access between the active *rootvg* and the alternative disk. The volume group that's woken up is renamed *altinst_rootvg*. The file systems of the volume group that is woken up are mounted under the mount point */alt_int*.

For instance, if the output of **lspv** is:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   old_rootvg
```

then after executing the command:

```
alt_disk_install -W hdisk1
```

the output of **lspv** becomes:

```
hdisk0     00008061bcd8a785   rootvg
hdisk1     000764abcd345678   altinst_rootvg
```

PUTTING A VOLUME GROUP TO SLEEP

```
    alt_disk_install -S
```

The **-S** flag lets the user put an alternative volume group that has been woken up to sleep. After executing the command, the alternative volume group is ‘varied off’ and all the */alt_inst* filesystems are unmounted. Note that this flag does not require any arguments (the operation is performed on the alternative volume group named *alt_inst_rootvg*) and that, after the execution of the command, the name of the alternative volume group becomes *altinst_rootvg*.

USING AN ALTERNATIVE DISK INSTALL

Being one of the newest additions to the operating system, the **alt_disk_install** command suffers from a few problems and issues concerning its installation and use. In this section I’ll try to provide some tips and warnings to help users.

The command ships in two filesets: *bos.alt_disk_install.rte* contains the **/usr/sbin/alt_disk_install** script and *bos.alt_disk_install.boot_images* contains the boot images required to install **mksysb** images to an alternative disk. The latest version of these filesets (currently 4.3.2.1) can be installed on AIX 4.1.4.0 and higher. Additionally, the fileset *bos.sysmgt.sysbr* must be installed on the system at the same level as *bos.rte*.

The *bos.alt_disk_install* package requires about 10 MB of disk space in the */usr* filesystem.

If the **mksysb** alternative disk install method is used, then the version level of **mksysb**’s installation must match that of the fileset *bos.alt_disk_install.boot_images*. Currently the following levels are supported: 4.3.0, 4.3.1, and 4.3.2.

Remember that the only safe method of removing an alternative volume group that’s supported is via the **alt_disk_install-X** command. If either the **exportvg** or the **reducevg** command is used, then a number of entries concerning essential filesystems are removed from the file */etc/filesystems* and the system is unable to reboot.

There must be sufficient free disk space in the root (/) filesystem when **alt_disk_install** is running, otherwise the command reports errors (but continues to process, finally aborting after creating an alternative disk that is unable to reboot the system).

If phased installation is performed, the user should ensure that no user process is using directories mounted under */alt_inst* before execution of the third phase (described above) begins, otherwise the command is unable to unmount the filesystem but continues to process, eventually creating an alternative disk with an illegal status.

As demonstrated by this article, the command has a large number of options. Therefore, I recommend users to make use of the appropriate Web-based System Manager (**wsm**) or **smit** menus (**smit alt_mksysb** for alternative disk **mksysb** install and **smit alt_clone** for alternative disk *rootvg* cloning).

REFERENCES

1 *AIX Version 4.3 Differences Guide*, SG24-2014-01, published by IBM, and also available from:

```
http://publib.boulder.ibm.com/pubs/pdfs/redbooks/sg242014.pdf
```

2 *AIX Version 4.3 Installation Guide*, SC23-4112, published by IBM, and also available from:

```
http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixins/
➤ aixinsgd/toc.htm
```

3 *AIX Version 4.3 Packaging Guide for LPP Installations*, published by IBM, and also available from:

```
http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixins/
➤ inslppkg/toc.htm
```

A Polak
System Engineer
APS (Israel)

Dynamic binding in AIX

Dynamic binding is to AIX what the DLL (Dynamic Link Library) is to Microsoft's Windows environment. Dynamic binding was introduced in AIX Version 3.1; it allows the linkage editor to delay resolving external symbol references until the application is executed or calls the *load()* subroutine to load an object file.

Dynamic binding falls into the two categories shown in Figure 1.

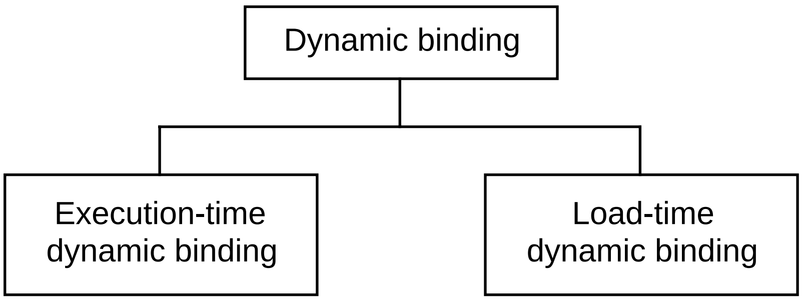

Figure 1: Types of dynamic binding

EXEC-TIME DYNAMIC BINDING

This allows the linkage editor to delay resolving external symbol references until the application is executed. For example, instead of copying the code for the *printf()* routine into the application, the linkage editor places a small amount of what is known as 'glue code' (see Figure 2) into the application. The glue code provides the name of the symbol (function or variable), and also the file name of the library or external module where the symbol is found. When the program is executed, the kernel's loader uses the glue code to locate and resolve the symbol. In the case of the *printf()* routine, the kernel's loader locates *printf()* in the standard C library and loads the code into the process image. The end result is a small executable.

SAMPLE.C

```
/*
 *  File containing the function Print ()
```

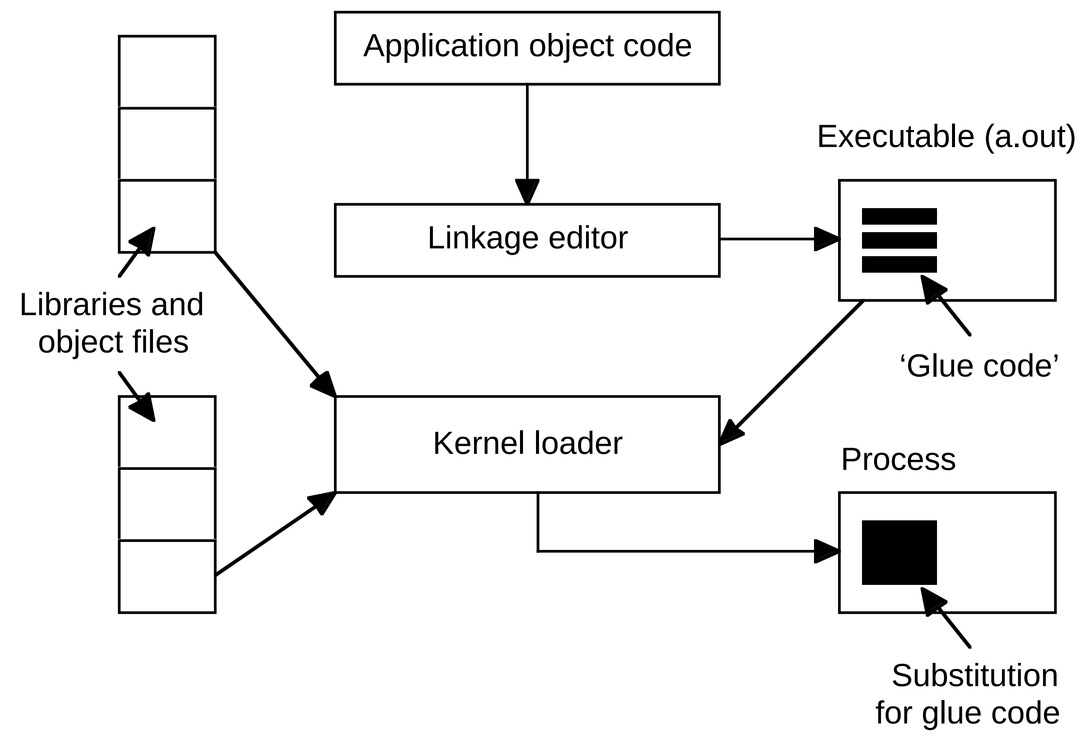

Figure 2: Glue code substitution

```
 */

Print ( )
{
    printf ( "Hello world\n" );
}
```

To create *sample.o,* the object file for *sample.c,* use the following command:

```
     cc -c sample.c
```

sample.exp is the ASCII export file required to produce object file for dynamic binding:

SAMPLE.EXP

```
# Export file for sample.c containing symbols to be made available
# for another executable object file to import

Print
```

Use the command line below to generate *sample*, a non-executable object file used for dynamic binding:

```
    ld  -o  sample  sample.o  -bE:sample.exp  -e  Print  -lc
```

In the above file:

- *b* is the flag for ‘binder option’.
- *E* is the flag for ‘export’.
- *e* flags the entry point function name, which is required by linkage editor.

testapp.c is the application that dynamically binds object file, *sample*:

TESTAPP.C

```
main ( )

{
/*
 *  Call the dynamically bound function
 */
    Print ( )
}
```

testapp.imp is an ASCII import file required to bind the object file, sample, dynamically:

TESTAPP.IMP

```
# Import file for testapp.c containing external symbols to be
# resolved at run time
#
# object file name with full path
#!./sample
# function name
Print
```

Use the command below to create **testapp**, the dynamically bound executable:

```
    cc  -o  testapp  testapp.c  -bI:testapp.imp
```

Where:

- *b* is the flag for ‘binder option’.
- *I* is the flag for ‘import’.

The size of **testapp** using dynamic binding is 2,194 bytes.

To create the **testapp** executable with static binding, use the command below:

```
cc  -o  testapp  testapp.c  sample.o
```

The size of **testapp** using static binding is 2,568 bytes.

ADVANTAGES

The two main advantages of using dynamic binding are that:

1 Executable files are smaller

2 Library routines can be changed without the need to relink applications to them.

DISADVANTAGE

The main disadvantages of using dynamic binding are that:

1 If for any reason a file referred to by glue code is not available, the loader fails when the user tries to execute the program.

2 To distribute applications that are dynamically bound to libraries or external modules, the libraries and modules must also be distributed.

LOAD-TIME DYNAMIC BINDING

This technique loads an object file into the process when the program running in the process calls the *load()* subroutine. Parameters passed to the *load()* subroutine include the name of the loadable object file and a flag option to control how the object file is loaded. The *load()* subroutine returns a pointer to the function that has been designated as the object file’s entry point. The object file’s code is loaded into the process’s data region. The *unload()* subroutine is called to unload the object file code from the data region when it is no longer needed.

LOAD() SYSTEM CALL

This loads and binds an object module into a current process. Unlike the **exec** system call, it does not replace the current program.

The calling syntax is:

```
#include <sys/ldr.h>
int ( * load ( char * Filepath, int Flags, char * LibraryPath ) )
```

Note that this function returns a pointer to a function.

UNLOAD() SYSTEM CALL

unload() unloads the object file and any imported object files that were automatically loaded with it. The pointer to the function returned by *load()* is passed to *unload()* as *FunctionPointer*.

The calling syntax is:

```
#include <sys/ldr.h>
int unload ( *FunctionPointer ( ) )
```

The *unload()* subroutine frees storage used by the specified object file only if the object file is no longer in use. An object file is in use as long as any other object file in use imports symbols from it.

When a library is unloaded, any unresolved symbol that is defined in the library is bound to the library's definition. These bindings create references to the library that cannot be undone even with the unload subroutine.

LISTING OF EXAMPLE

Below is the source code, *bigfundc.c,* which contains definitions for two functions, *Bigfunc1()* and *Bigfunc2().* These are meant to be large and only used occasionally by the application.

BIGFUNC.C

```
/*
 *  File containing the functions Bigfunc1 ( ) and BigFunc2 ( )
 */
void Bigfunc1 ( )
{
    printf( "This is big function 1, which is hardly used.\n" );
}

void Bigfunc2 ( )
{
    printf( "This is big function 2, which is hardly used.\n" );
}
```

Use the command below to obtain *bigfunc.o,* the simple object file for *bigfunc.c*.

```
     cc  -c  bigfunc.c
```

To generate *bigfunc*, a non-executable object file for dynamic binding, use the command below.

```
     ld  -o  bigfunc  sample.o  -e  Bigfunc2  -lc
```

The **e** flag is used to specify the entry point function name (that is, the function that is to be loaded dynamically).

Below is the code for *testapp.c*, an application that dynamically loads function *Bigfunc2()*.

TESTAPP.C

```
char  *libpath=".";     /*  must be a directory  */

main ( )
{
    /*
     * declare a pointer to function
     */
    int (*funcp) ( );

    /*
     * load the required Bigfunc2 ( ) from file bigfunc
     */
    if ( ( funcp = load( "bigfunc", 0, libpath ) ) == 0 )
    {
        perror( "load" );
        exit ( 1 );
    }

    /*
     * execute the function
     */
    ( *funcp ) ( );
    /*
     *  unload   the  function
     */
    unload ( funcp );
}
```

Use the command below to create the executable **testapp**.

```
    cc  -o  testapp  testapp.c
```

The advantage of using this method is that loading the large function on-the-fly, instead of when the program is executed, reduces virtual memory requirements of the process.

The disadvantage of using it is that there is a danger that the file that holds required functions will be absent.

Arif Zaman
DBA/Developer
High-Tech Software Ltd (UK)

Is AIX ready for the year 2000?

IBM states that: "A product is Year 2000 ready if, when used in accordance with its associated documentation, it is capable of correctly processing, providing, and/or receiving date data within and between the 20th and 21st centuries, provided all other products (for example, software, hardware, and firmware) used with the product properly exchange accurate date data with it." But is AIX Year 2000 ready?

INFORMATION ABOUT AIX AND THE YEAR 2000

IBM offers complete information about AIX's Year 2000 readiness in the document '*AIX, Unix Operating Systems, and the Year 2000 Issue: An Informational Workbook*'. This is available at: *http://www.software.ibm.com/year2000/papers/aixy2k.html*.

The document deals directly with the issues as they relate to AIX, Unix, and RS/6000-related hardware. It covers:

- How Unix-based operating systems are affected by the Year 2000 problem.
- Readiness information for all releases of AIX.
- Readiness information for RS/6000 hardware.
- Patches available for earlier releases of AIX.
- Test planning guide for software.

The intent is to provide the software developer, administrator, or end-user with the information required to make informed decisions when dealing with Year 2000 and related issues. It cannot provide specific solutions to all problems that may be encountered in this area, but should lead the reader to further sources of information, in the event that the information in this document is not sufficient.

SOFTWARE TOOL CHECKS FOR YEAR 2000 READINESS

New Year 2000 issues in AIX were recently reported that impact all Year 2000-ready versions of the AIX operating system. None of the problems, however, is expected to impact internal data integrity. Problems occur in specific AIX commands and in certain programs provided with AIX. They appear in less frequently used parts of AIX and in some sample programs provided with AIX. Nevertheless, they will require corrective fixes.

To find out whether an AIX installation has the correct updates, IBM provides a downloadable tool that runs on AIX 3.2.5, and AIX 4.1, 4.2, or 4.3 systems. The tool determines whether all the necessary fixes have been applied so that the system complies with the latest known state of Year 2000 readiness. The tool is a binary executable and does not require root permission to run. The file is **aixy2k_fixcheck18** (as of September 1998) and it can be downloaded from: *ftp://ftp.software.ibm.com/aix/tools/y2k.*

Here's its output on my system:

```
$ ./aixY2K_check18

******** AIX Year 2000 Update Search Tool ******
(v1.8 August 1998)

Starting check. Getting version information.
AIX Version 4 Release 2
Searching update database........

...

***********  RESULTS OF UPDATE SEARCH **********

This installation DOES NOT have all the current
Year 2000 updates for this release.

IBM recommends that you order and apply
          APAR IX81294
in order to bring this installation to Year 2000
Ready status.

IBM recommends that you regularly check its
Year 2000 information center for updated status
at http://www.ibm.com/IBM/year2000/ .

$
```

AIX 4.2.1 and AIX 4.3, which were previously listed as Year 2000 ready without PTFs, are now considered Year 2000 ready only after the appropriate PTFs are applied. All other AIX releases that have been listed as Year 2000 ready with the application of specific PTFs now require additional PTFs.

A list of the latest fixes for AIX releases is available on the Web at: *http://www.software.ibm.com/year2000/papers/aixy2k.html*. For further information about which PTFs are needed for which AIX release, visit their 'AIX Problem Solving Database Web site' at *http://www.rs6000.ibm.com/solutions/y2k/*.

It is more than likely that today's fixes will be replaced by newer ones. Frankly, it would not surprise me if some of the Year 2000 readiness fixes were not available until after January 1st, 2000!

Werner Klauser
Klauser Informatik (Switzerland)

Systems monitoring shell script (part 2)

This month's instalment concludes this article on a systems monitoring shell script.

CHECKITALL (CONTINUED)

```
Home_Check()
{
  typeset -i HOMECNT=$HOMECNT
  d="/dev/hd1 (home)"
  USED=`df /dev/hd1|grep -v Filesystem|awk '{print $4}'|awk -F%
  ➤  '{print $1}'`
  if [ $USED -ge $HOME_MAX ]
    then
      echo "$d has reached $USED% utilization."  >> $LOGFILE
      GOOD=1
      if [ $HOMECNT = 0 ]
        then
          echo "Mailing system administration..."  >> $LOGFILE
          HOMECNT=$HOMECNT+1
          Mail C $d $USED
        else
          if [ $HOMECNT -ge $HOMELIMIT ]
            then
              HOMECNT=0
            else
              HOMECNT=$HOMECNT+1
          fi
      fi
    else
      HOMECNT=0
  fi
  echo "HOMECNT $HOMECNT"  >> /tmp/Counters
}

######################################################################
#
# U01_Check()
#
#   U01_Check() checks the amount of space used by individual
#   user-defined filesystems. If this exceeds MAX, and the counter
#   is set to zero, an alert is sent.
#
######################################################################
U01_Check()
```

```
{
  typeset -i U01CNT=$U01CNT
  d="/dev/u01lv (U01)"
  df /dev/u01lv 1>/dev/null 2>/dev/null
  if [ $? -ne 0 ]
    then
      Mail K $d
    else
      USED=`df /dev/u01lv|grep -v Filesys|awk '{print $4}'|awk -F%
      ➤  '{print $1}'`
        if [ $USED -ge $U01_MAX ]
          then
            echo "$d has reached $USED% utilization."  >> $LOGFILE
            GOOD=1
            if [ $U01CNT = 0 ]
              then
                echo "Mailing system administration..."  >> $LOGFILE
                U01CNT=$U01CNT+1
                Mail C $d $USED
              else
                if [ $U01CNT -ge $U01LIMIT ]
                  then
                    U01CNT=0
                  else
                    U01CNT=$U01CNT+1
                fi
            fi
          else
            U01CNT=0
        fi
  fi
  echo "U01CNT $U01CNT"  >> /tmp/Counters
}

#####################################################################
#
# CPU_Load()
#
#   CPU_Load() establishes the average load on the system during
#   the previous five minute interval. The average load is obtained
#   using the 'uptime' command. If the average load exceeds CPU_MAX,
#   and the counter is set to zero, an alert is sent.
#
#####################################################################
CPU_Load()
{
  typeset -i CPUCNT=$CPUCNT
  for c in `uptime | awk '{print $10}'`
  do
    if [ $c = "average:" ]
```

```
      then
        c=`uptime | awk '{print $11}'`
    fi
    cpload=`echo $c | awk -F. '{print $1}'`
    if [ $cpload -ge $CPU_MAX ]
      then
        echo "The CPU Load has reached $c."  >> $LOGFILE
        GOOD=1
        if [ $CPUCNT = 0 ]
          then
            echo "Mailing system administration..."  >> $LOGFILE
            CPUCNT=$CPUCNT+1
            Mail D $c
          else
            if [ $CPUCNT -ge $CPULIMIT ]
              then
                CPUCNT=0
              else
                CPUCNT=$CPUCNT+1
            fi
        fi
      else
        CPUCNT=0
    fi
  done
  echo "CPUCNT $CPUCNT"  >> /tmp/Counters
}

#####################################################################
#
# DB_Check()
#
#   DB_Check() checks the availability of a DB2 database. If the
#   database is not available, and the counter is set to zero, an
#   alert is sent.
#
#   DB_Check() is set up not to check for database availability on
#   Sundays until 4:00pm. This is a requirement of ours to allow
#   time to carry out such tasks as reorganizing and backing up
#   the database.
#
#   Note also that this function is specific to DB2 for AIX. By
#   changing the method of connecting to the database, it can be
#   made compatible with Oracle, Sybase, Informix, etc. If used
#   with DB2 for AIX, please alter the connect statement to refer
#   to the correct 'INSTANCE'.
#
#####################################################################
DB_Check()
{
```

```
typeset -i DB2CNT=$DB2CNT
DAY=`date | awk '{print $1}'`
if [ $DAY = "Sun" ]
  then
    HOUR=`date | awk '{print $4}' | awk -F: '{print $1}'`
    if [ $HOUR -ge 16 ]
      then
        /bin/su - db2 -c db2 connect to INSTANCE 1> /dev/null 2>
        ➤  /dev/null
        if [ $? != 0 ]
          then
            echo "An attempted database connection has failed."
            ➤  >> $LOGFILE
            GOOD=1
            if [ $DB2CNT = 0 ]
              then
                echo "Mailing system administration..."  >> $LOGFILE
                DB2CNT=$DB2CNT+1
                Mail E
              else
                if [ $DB2CNT -ge $DB2LIMIT ]
                  then
                    DB2CNT=0
                  else
                    DB2CNT=DB2CNT+1
                fi
            fi
          else
            DBCNT=0
        fi
    fi
  else
    /bin/su - db2 -c db2 connect to INSTANCE 1> /dev/null 2>
    ➤  /dev/null
    if [ $? != 0 ]
      then
        echo "An attempted database connection has failed."
        ➤  >> $LOGFILE
        GOOD=1
        if [ $DB2CNT = 0 ]
          then
            echo "Mailing system administration..."  >> $LOGFILE
            DB2CNT=$DB2CNT+1
            Mail E
          else
            if [ $DB2CNT -ge $DB2LIMIT ]
              then
                DB2CNT=0
              else
                DB2CNT=DB2CNT+1
            fi
          fi
            else
              DBCNT=0
          fi
  fi
  /bin/su - db2 -c db2 connect reset 1> /dev/null 2> /dev/null
  echo "DB2CNT $DB2CNT"  >> /tmp/Counters
}

#######################################################################
#
# SNA_Check()
#
#   SNA_Check() checks the status of SNA connectivity to a remote
#   host. If the SNA link 'LINK' is not 'active', and the counter
#   is set to zero, an alert is sent. As with DB_Check(),
#   SNA_Check() does not check SNA connectivity on Sundays until
#   after 4:00pm. This accommodates the scheduled weekly outage of
#   our remote host.
#
#   SNA_Check() needs to have the statement 'sna -d l' updated to
#   reflect the 'grep LINKNAME' of the appropriate SNA link
#   station name.
#
#   After the status of the link station is determined as active,
#   the number of active sessions on the link station is evaluated.
#   If the number of active sessions is below the ACTIVESNA_MIN
#   global variable, and ACTIVESNA_CNT is zero, an alert is sent.
#
#######################################################################
SNA_Check()
{
  typeset -i SNACNT=$SNACNT
  DAY=`date | awk '{print $1}'`
  if [ $DAY = "Sun" ]
    then
      HOUR=`date | awk '{print $4}' | awk -F: '{print $1}'`
      if [ $HOUR -ge 16 ]
        then
          SNAstatus=`/usr/bin/sna -d l | grep LINK | awk '{print $5}'`
          if [ $SNAstatus != "Active" ]
            then
              echo "SNA link LINK does not appear to be active."
              ➤  >> $LOGFILE
              GOOD=1
              if [ $SNACNT = 0 ]
                then
                  echo "Mailing system administration."  >> $LOGFILE
                  SNACNT=$SNACNT+1
                  Mail F
                else
                  if [ $SNACNT -ge $SNALIMIT ]
                    then
                      SNACNT=0
                    else
                      SNACNT=$SNACNT+1
                  fi
              fi
            else
              typeset -i ACTIVESESSIONS=`sna -d l|grep LINK|awk
              ➤  '{print $6}'`
              typeset -i ACTIVESNA_NEWMIN=$ACTIVESNA_MIN-1
              typeset -i ACTIVESNA_CNT=$ACTIVESNA_CNT
                if [ $ACTIVESESSIONS -le $ACTIVESNA_NEWMIN ]
                  then
                    echo "There are $ACTIVESESSIONS active sessions."
                    ➤  >> $LOGFILE
                    GOOD=1
                      if [ $ACTIVESNA_CNT = 0 ]
                        then
                          echo "Mailing system administration."
                          ➤  >> $LOGFILE
                          ACTIVESNA_CNT=$ACTIVESNA_CNT+1
                          Mail U $ACTIVESESSIONS
                        else
                          if [ $ACTIVESNA_CNT -ge $ACTIVESNALIMIT ]
                            then
                              ACTIVESNA_CNT=0
                            else
                              ACTIVESNA_CNT=$ACTIVESNA_CNT+1
                          fi
                      fi
                  else
                    ACTIVESNA_CNT=0
                fi
          fi
      fi
    else
      SNAstatus=`/usr/bin/sna -d l | grep LINK | awk '{print $5}'`
      if [ $SNAstatus != "Active" ]
        then
          echo "SNA link LINK does not appear to be active."
          ➤  >> $LOGFILE
          GOOD=1
          if [ $SNACNT = 0 ]
            then
              echo "Mailing system administration."  >> $LOGFILE
              SNACNT=$SNACNT+1
              Mail F
```

```
            else
              if [ $SNACNT -ge $SNALIMIT ]
                then
                  SNACNT=0
                else
                  SNACNT=$SNACNT+1
              fi
          fi
        else
          typeset -i ACTIVESESSIONS=`sna -d l|grep LINK|awk
          ➤  '{print $6}'`
          typeset -i ACTIVESNA_NEWMIN=$ACTIVESNA_MIN-1
          typeset -i ACTIVESNA_CNT=$ACTIVESNA_CNT
            if [ $ACTIVESESSIONS -le $ACTIVESNA_NEWMIN ]
              then
                echo "There are $ACTIVESESSIONS active sessions."
                ➤  >> $LOGFILE
                GOOD=1
                  if [ $ACTIVESNA_CNT = 0 ]
                    then
                      echo "Mailing system administration."
                      ➤  >> $LOGFILE
                      ACTIVESNA_CNT=$ACTIVESNA_CNT+1
                      Mail U $ACTIVESESSIONS
                    else
                      if [ $ACTIVESNA_CNT -ge $ACTIVESNALIMIT ]
                        then
                          ACTIVESNA_CNT=0
                        else
                          ACTIVESNA_CNT=$ACTIVESNA_CNT+1
                      fi
                  fi
              else
                ACTIVESNA_CNT=0
            fi
      fi
  fi
  echo "SNACNT $SNACNT"  >> /tmp/Counters
  echo "ACTIVESNA_CNT $ACTIVESNA_CNT"  >> /tmp/Counters
}

#####################################################################
#
# Offline()
#
#   Offline() is used to determine whether a physical volume is
#   in a 'defined' state. If the physical volume is in this state,
#   and the counter is set to zero, an alert is sent.
#
#####################################################################
```

```
Offline()
{
  typeset -i OFFDISKCNT=$OFFDISKCNT
  for d in `lsdev -Cc disk | grep Defined`
    do
      echo "A disk drive is in a defined state."  >> $LOGFILE
      GOOD=1
      if [ $OFFDISKCNT = 0 ]
        then
          echo "Mailing system administration..."  >> $LOGFILE
          OFFDISKCNT=$OFFDISKCNT+1
          Mail H
        else
          if [ $OFFDISKCNT -ge $OFFDISKLIMIT ]
            then
              OFFDISKCNT=0
            else
              OFFDISKCNT=$OFFDISKCNT+1
          fi
      fi
    done
  echo "OFFDISKCNT  $OFFDISKCNT"  >> /tmp/Counters
}

#####################################################################
#
# PV2VG()
#
#   PV2VG() is used to verify that each physical volume in the
#   system is assigned to a volume group. If a physical volume is
#   not assigned to a volume group, and the counter is set to
#   zero, an alert is sent.
#
#####################################################################
PV2VG()
{
  typeset -i PV2VGCNT=$PV2VGCNT
  for v in `lspv | grep None | awk '{print $1}'`
    do
      echo "$v is not assigned to a volume group."  >> $LOGFILE
      GOOD=1
      if [ $PV2VGCNT = 0 ]
        then
          echo "Mailing system administration."  >> $LOGFILE
          Mail I $v
          PV2VGCNT=$PV2VGCNT+1
        else
          if [ $PV2VGCNT -ge $PV2VGLIMIT ]
            then
              PV2VGCNT=0
          else
            PV2VGCNT=$PV2VGCNT+1
        fi
    fi
  done
 echo "PV2VGCNT $PV2VGCNT"  >> /tmp/Counters
}

###############################################################################
#
# Active_PROC()
#
#   Active_PROC() monitors the status of a particular process
#   that is expected to be active at all times. The 'grep PROC'
#   and 'echo' statements need to be updated to reflect the name
#   of the process to be monitored. The 'Mail' statement also
#   needs updating to reflect the name of the process.
#
#   If the process is not active when the script is run, and
#   the counter is set to zero, an alert is sent.
#
###############################################################################
Active_PROC()
{
  typeset -i PROCCNT=$PROCCNT
  ps -ef | grep PROC | grep -v grep > /dev/null
  if [ $? != 0 ]
    then
      echo "PROC does not appear to be active."  >> $LOGFILE
      GOOD=1
      if [ $PROCCNT = 0 ]
        then
          echo "Mailing system administration..."  >> $LOGFILE
          Mail J PROC
          PROCCNT=$PROCCNT+1
        else
          if [ $PROCCNT -ge $PROCLIMIT ]
            then
              PROCCNT=0
            else
              PROCCNT=$PROCCNT+1
          fi
      fi
    else
      PROCCNT=0
  fi
  echo "PROCCNT $PROCCNT"  >> /tmp/Counters
}

###############################################################################
```

```
#
# Active_QDAEMON()
#
#   Active_QDAEMON() is similar to Active_PROC() in that it
#   monitors the status of the qdaemon. If the qdaemon is not
#   active, and the counter is set to zero, an alert is sent.
#
######################################################################
Active_QDAEMON()
{
  typeset -i QDAEMONCNT=$QDAEMONCNT
  ps -ef | grep qdaemon | grep -v grep > /dev/null
  if [ $? != 0 ]
    then
      echo "qdaemon does not appear to be active."  >> $LOGFILE
      GOOD=1
      if [ $QDAEMONCNT = 0 ]
        then
          echo "Mailing system administration..."  >> $LOGFILE
          Mail J qdaemon
          QDAEMONCNT=$QDAEMONCNT+1
        else
          if [ $QDAEMONCNT -ge $QDAEMONLIMIT ]
            then
              QDAEMONCNT=0
            else
              QDAEMONCNT=$QDAEMONCNT+1
          fi
      fi
    else
      QDAEMONCNT=0
  fi
  echo "QDAEMONCNT $QDAEMONCNT"  >> /tmp/Counters
}

######################################################################
#
# UPTIME()
#
#   UPTIME() is used to monitor the time since last system reboot.
#   If the number of days since the last reboot exceeds UPTIME_MAX,
#   and the counter is set to zero, an alert is sent. This function
#   is useful if systems are set to reboot automatically on a
#   periodic basis.
#
######################################################################
UPTIME()
{
  TIMER=`uptime | awk '{print $4}' | awk -F, '{print $1}'`
  if [ $TIMER = "days" ]
```

```
    then
      typeset -i UPTIMECNT=$UPTIMECNT
      UpTime=`uptime | awk '{print $3}' | awk -F: '{print $1}'`
      if [ $UpTime -ge $UPTIME_MAX ]
        then
          echo "The system has not been rebooted for $UpTime days."
          ➤  >> $LOGFILE
          GOOD=1
          if [ $UPTIMECNT = 0 ]
            then
              echo "Mailing system administration..."  >> $LOGFILE
              Mail L $UpTime
              UPTIMECNT=$UPTIMECNT+1
            else
              if [ $UPTIMECNT -ge $UPTIMELIMIT ]
                then
                  UPTIMECNT=0
                else
                  UPTIMECNT=$UPTIMECNT+1
              fi
          fi
        else
          UPTIMECNT=0
      fi
    else
      if [ $TIMER = "day" ]
        then
          UPTIMECNT=0
      fi
  fi
  echo "UPTIMECNT $UPTIMECNT"  >> /tmp/Counters
}

#####################################################################
#
# 24Hrs_Ago()
#
#   24Hrs_Ago() is a crude, though effective, way of obtaining
#   the date and time stamp of a period exactly 24 hour ago. It
#   takes into account leap years up to the year 2040. The
#   function defines the variable 24HRSAGO, which is used by
#   ERRORLOG.
#
#####################################################################
24Hrs_Ago()
{
  CUR_MONTH=`date +%m`
  CUR_TIME=`date +%H%M`
  typeset -i CUR_MONTH1=`date +%m`
  typeset -i CUR_DAY=`date +%d`
  typeset -i CUR_YEAR=`date +%y`
  typeset -i CUR_JULE=`date +%j`
  typeset -i NEW_MONTH
  typeset -i NEW_DAY
  typeset -i NEW_YEAR
  typeset -i NEW_JULE
  if [ $CUR_JULE = 1 ]
    then
      if [ $CUR_YEAR = 0 ]
        then
          NEW_YEAR=99
        else
          NEW_YEAR=$CUR_YEAR-1
      fi
    else
      NEW_YEAR=$CUR_YEAR
  fi
  case $CUR_MONTH in
    01)
       if [ $CUR_DAY = 1 ]
         then
           NEW_DAY=31
           NEW_MONTH=12
         else
           NEW_DAY=$CUR_DAY-1
           NEW_MONTH=$CUR_MONTH
       fi
       ;;
    02|04|06|08|09|11)
       if [ $CUR_DAY = 1 ]
         then
           NEW_DAY=31
           NEW_MONTH=$CUR_MONTH1-1
         else
           NEW_DAY=$CUR_DAY-1
           NEW_MONTH=$CUR_MONTH
       fi
       ;;
    03)
       if [ $CUR_DAY = 1 ]
         then
           CUR_YEARFULL=`date +%Y`
           case $CUR_YEARFULL in
             2000|2004|2008|2012|2016|2020|2024|2028|2032|2036)
               NEW_DAY=29
               ;;
             *)
               NEW_DAY=28
               ;;
           esac
```

```
            NEW_MONTH=02
          else
            NEW_DAY=$CUR_DAY-1
            NEW_MONTH=$CUR_MONTH
        fi
        ;;
    05|07|10|12)
        if [ $CUR_DAY = 1 ]
          then
            NEW_DAY=30
            NEW_MONTH=$CUR_MONTH1-1
          else
            NEW_DAY=$CUR_DAY-1
            NEW_MONTH=$CUR_MONTH
        fi
        ;;
      *)
        echo "Unknown month."
        ;;
  esac
  typeset -i DAY_LENGTH=`expr length $NEW_DAY`
  typeset -i MONTH_LENGTH=`expr length $NEW_MONTH`
  if [ $DAY_LENGTH = 1 ]
    then
      DAY_OUT="0$NEW_DAY"
    else
      DAY_OUT=$NEW_DAY
  fi
  if [ $MONTH_LENGTH = 1 ]
    then
      MONTH_OUT="0$NEW_MONTH"
    else
      MONTH_OUT=$NEW_MONTH
  fi
  export 24HRSAGO=`echo $MONTH_OUT$DAY_OUT$CUR_TIME$NEW_YEAR`
}

######################################################################
#
# ERRORLOG()
#
#   ERRORLOG() takes the 24HRSAGO variable obtained from the
#   function 24Hrs_Ago() and checks the system errorlog for the
#   previous 24 hour period. The entire error log is written to a
#   temporary file, and then the file /tmp/greperrors is
#   referenced. Any error code ID in the greperrors file is
#   ignored. If there are errors to report after processing
#   greperror, and the ERRORLOGCNT variable is set to zero, an
#   alert is sent.
#
```

```
#   To build the /tmp/greperrors file, redirect the output of
#   'errpt -t' to /tmp/greperrors. The only data needed in
#   greperrors is a column of error IDs you wish to be omitted
#   by ERRORLOG(). Also include an 'IDENTIFIER' entry in
#   /tmp/greperrors to prevent errpt's header being reported
#   as an error.
#
######################################################################
ERRORLOG()
{
  typeset -i ERRORLOGCNT=$ERRORLOGCNT
  cat /dev/null > /tmp/errorlog.checkitall
  errpt -s $24HRSAGO > /tmp/errorlog.checkitall
  for e in `cat /usr/local/bin/greperrors`
  do
    cat /tmp/errorlog.checkitall | grep -v $e >
    ➤  /tmp/errorlog1.checkitall
    mv /tmp/errorlog1.checkitall /tmp/errorlog.checkitall
  done
  typeset -i FILELENGTH=`wc -c /tmp/errorlog.checkitall | awk
  ➤  '{print $1}'`
  if [ $FILELENGTH != 0 ]
    then
      echo "The error log has reported new error messages."
      ➤  >> $LOGFILE
      GOOD=1
      if [ $ERRORLOGCNT = 0 ]
        then
          echo "Mailing system administration."  >> $LOGFILE
          Mail P
          ERRORLOGCNT=$ERRORLOGCNT+1
        else
          if [ $ERRORLOGCNT -ge $ERRORLOGLIMIT ]
            then
              ERRORLOGCNT=0
            else
              ERRORLOGCNT=$ERRORLOGCNT+1
          fi
      fi
  fi
  echo "ERRORLOGCNT $ERRORLOGCNT"  >> /tmp/Counters
}

######################################################################
#
# QuorumCheck()
#
#   QuorumCheck() checks the status of quorum on each volume group
#   defined to the system. In a mirrored environment, it is
#   essential to have quorum disabled. If your system is mirrored,
#   the QuorumCheck function verifies that quorum is disabled for
#   your volume groups. If you need quorum turned on, change '1'
#   to '2' in the statement 'if [ $QUORUMSTATUS != 1 ]'. This
#   verifies that quorum is turned on.
#
######################################################################
QuorumCheck()
{
  typeset -i QUORUMCNT=$QUORUMCNT
  for v in `lsvg`
    do
      typeset -i QUORUMStatus=`lsvg $v | grep QUORUM | awk
      ➤  '{print $5}'`
        if [ $QUORUMStatus != 1 ]
          then
            echo "$v appears to have Quorum turned on."  >> $LOGFILE
            GOOD=1
              if [ $QUORUMCNT = 0 ]
                then
                  echo "Mailing system administration."  >> $LOGFILE
                  Mail Q $v
                  QUORUMCNT=$QUORUMCNT+1
                else
                  if [ $QUORUMCNT -ge $QUORUMLIMIT ]
                    then
                      QUORUMCNT=0
                    else
                      QUORUMCNT=$QUORUMCNT+1
                  fi
              fi
          else
            QUORUMCNT=0
        fi
    done
  echo "QUORUMCNT $QUORUMCNT"  >> /tmp/Counters
}

######################################################################
#
# MirrorCheck()
#
#   MirrorCheck() checks the status of logical volume mirrors.
#   The number of physical partitions is monitored to verify
#   that copies on separate physical volumes are balanced. The
#   mirrors are also verified to ensure that each physical volume
#   in the mirror has the same number of logical partitions. If
#   an error is encountered in the mirroring set-up, and the
#   counter is set to zero, an alert is sent.
#
######################################################################
```

```
MirrorCheck()
{
  typeset -i MIRRORSAMEPVCNT=$MIRRORSAMEPVCNT
  typeset -i MIRRORCNTPVCNT=$MIRRORCNTPVCNT
  typeset -i MIRRORPVOFFCNT=$MIRRORPVOFFCNT
  for m in `lsvg -o|lsvg -i -l|grep -v vg|grep -v TYPE|grep -v hd7| \
     awk '{print $1}'`
    do
      WRONGPVCopies=0
      lslv -m $m | grep -v $m | grep -v PP1 > /tmp/lslv.checkitall
      PV1=`cat /tmp/lslv.checkitall | head -1 | awk '{print $3}'`
      PV2=`cat /tmp/lslv.checkitall | head -1 | awk '{print $5}'`
        if [ $PV1 = $PV2 ]
          then
            echo "$m appears to have its copies on the same disk."
            ➤  >> $LOGFILE
            GOOD=1
              if [ $MIRRORSAMEPVCNT = 0 ]
                then
                  echo "Mailing system administration."  >> $LOGFILE
                  Mail R $m
                  MIRRORSAMEPVCNT=$MIRRORSAMEPVCNT+1
                else
                  if [ $MIRRORSAMEPVCNT -ge $MIRRORSAMEPVLIMIT ]
                    then
                      MIRRORSAMEPVCNT=0
                    else
                      MIRRORSAMEPVCNT=$MIRRORSAMEPVCNT+1
                  fi
              fi
          else
            MIRRORSAMEPVCNT=0
        fi
      PV1Count=`cat /tmp/lslv.checkitall | grep $PV1 | wc -l`
      PV2Count=`cat /tmp/lslv.checkitall | grep $PV2 | wc -l`
        if [ $PV1Count != $PV2Count ]
          then
            echo "$m does not have the same LP count on its PVs."
            ➤  >> $LOGFILE
            GOOD=1
              if [ $MIRRORCNTPVCNT = 0 ]
                then
                  echo "Mailing system administration."  >> $LOGFILE
                  Mail S $m
                  MIRRORCNTPVCNT=$MIRRORCNTPVCNT+1
                else
                  if [ $MIRRORCNTPVCNT -ge $MIRRORCNTPVLIMIT ]
                    then
                      MIRRORCNTPVCNT=0
                    else
                      MIRRORCNTPVCNT=$MIRRORCNTPVCNT+1
                  fi
              fi
          else
            MIRRORCNTPVCNT=0
        fi
      for c in `cat /tmp/lslv.checkitall | grep $PV1 | awk
      ➤  '{print $3}'`
        do
          if [ $c != $PV1 ]
            then
              WRONGPVCopies=1
          fi
        done
      for c in `cat /tmp/lslv.checkitall | grep $PV2 | awk
      ➤  '{print $5}'`
        do
          if [ $c != $PV2 ]
            then
              WRONGPVCopies=1
          fi
        done
      if [ $WRONGPVCopies = 1 ]
        then
          echo "$m appears to have single copies on multiple PVs."
          ➤  >> $LOGFILE
          GOOD=1
            if [ $MIRRORPVOFFCNT = 0 ]
              then
                echo "Mailing system administration."  >> $LOGFILE
                Mail T $m
                MIRRORPVOFFCNT=$MIRRORPVOFFCNT+1
              else
                if [ $MIRRORPVOFFCNT -ge $MIRRORPVOFFLIMIT ]
                  then
                    MIRRORPVOFFCNT=0
                  else
                    MIRRORPVOFFCNT=$MIRRORPVOFFCNT+1
                fi
            fi
        else
          MIRRORPVOFFCNT=0
      fi
      rm -f /tmp/lslv.checkitall
    done
  echo "MIRRORSAMEPVCNT $MIRRORSAMEPVCNT"  >> /tmp/Counters
  echo "MIRRORCNTPVCNT $MIRRORCNTPVCNT"    >> /tmp/Counters
  echo "MIRRORPVOFFCNT $MIRRORPVOFFCNT"    >> /tmp/Counters
}
```

```
#########################################################################
#
# All_Good()
#
#   All_Good() is the last function to be called. Throughout the
#   script, if an error is encountered, the GOOD global variable
#   is set to '1'. If GOOD is '0' at the end of the run, then no
#   error has been encountered, and the LOGFILE is updated
#   accordingly.
#
#########################################################################
All_Good()
{
  if [ $GOOD = 0 ]
    then
      echo "No warnings/errors encountered..."  >> $LOGFILE
  fi
}

#########################################################################
#
# Mail()
#
#   Mail() is called by other functions in the event that an error
#   is encountered and the counter for that particular error is
#   set to zero.
#
#   Alert notification is carried out by the Mail function.
#   Various parameters are passed to Mail. For instance, if the
#   root filesystem has reached the threshold specified by
#   ROOT_MAX, the Mail function is called by:
#
#     Mail C /dev/hd4 (root) 98
#
#   This directs Mail to use text 'C' in the case statement that
#   follows, and so is passed the text: "/dev/hd4 (root) has
#   reached 98% on HOST". After the TEXT variable is set to the
#   appropriate value, TEXT is mailed to the specified ID. If
#   it's necessary to direct mail to more than one user, I
#   suggest mailing to a single Unix user ID created specifically
#   for the script. Then create a ".forward" file that directs
#   mail from this ID to other user IDs.
#
#   I've also found that an Internet mail ID can be included in
#   the .forward file to enable e-mail messages to be forwarded
#   to alphanumeric pagers. A word of caution: alphanumeric
#   pagers can get flooded with pages if you try to send lenghty
#   messages (such as the error report). I suggest mailing the
#   error report to a user ID that does not forward to a pager.
#
```

```
################################################################################
Mail()
{
  if [ $# -eq 0 ]
    then
      echo "The MAIL function was called with no arguments."
      ➤  >> $LOGFILE
    else
      case $1 in
        A)
          TEXT="$HOST has reported $2 stale partitions on $3."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        B)
          TEXT="Paging space on $HOST has reached $2%."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        C)
          TEXT="$2 $3 has reached $4% on $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        D)
          TEXT="CPU Load on $HOST has reached $2."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        E)
          TEXT="An attempted database connection on $HOST has failed."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        F)
          TEXT="The SNA link LINK does not appear to be active on
          ➤  $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        H)
          TEXT="A disk drive is in a defined state on $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        I)
          TEXT="$2 is not assigned to a volume group."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        J)
          TEXT="The $2 process does not appear to be active on $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        K)
          TEXT="$2 does not appear to be mounted on $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        L)
          TEXT="$HOST has not been rebooted for $2 days."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        P)
          echo "$HOST has the following errors: "
          ➤  > /tmp/checkitall.mail
          cat /tmp/errorlog.checkitall >> /tmp/checkitall.mail
          mail daffy@warnerbros.com < /tmp/checkitall.mail
          rm -f /tmp/checkitall.mail
          ;;
        Q)
          TEXT="$2 appears to have Quorum turned on at $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        R)
          TEXT="$2 has multiple copies on the same disk."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        S)
          TEXT="$2 does not have the same LP count on its physical
          ➤  volumes."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        T)
          TEXT="$2 has single copies on multiple physical volumes."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        U)
          TEXT="There are $2 active SNA sessions on $HOST."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
        *)
          TEXT="An unknown mail argument has been passed."
          echo $TEXT | mail daffy@warnerbros.com
          ;;
      esac
  fi
}

######################################################################
#
# Main section
#
#   This section simply calls the functions defined above.
#
#   Note the 'if' statement, which checks the value of
#   DEFAULTCHECKCNT. This check was put in place so that the
#   ERRORLOG(), QuorumCheck(), and MirrorCheck() are called only
#   every DEFAULTCHECKLIMIT times. In this example these functions
#   are called only after the script has run 48 times. This
#   prevents CPU-intensive functions, such as those that
#   manipulate the error log and query the ODM, from pounding
#   the CPU every five minutes (or however often you choose to
#   call the script from crontab). If you call the script only
#   every hour or so, you might want to remove this test and
#   simply call all functions each time the script is run.
#
######################################################################
LogTime
Read_Counters
Stale
Paging
Root_Check
Var_Check
Tmp_Check
Home_Check
U01_Check
CPU_Load
DB_Check
SNA_Check
Offline
PV2VG
Active_PROC
Active_CRON
Active_QDAEMON
UPTIME

typeset -i DEFAULTCHECKCNT=$DEFAULTCHECKCNT
if [ $DEFAULTCHECKCNT = 0 ]
  then
    24Hrs_Ago
    ERRORLOG
    QuorumCheck
    MirrorCheck
    DEFAULTCHECKCNT=$DEFAULTCHECKCNT+1
  else
    if [ $DEFAULTCHECKCNT -ge $DEFAULTCHECKLIMIT ]
      then
        DEFAULTCHECKCNT=0
      else
        DEFAULTCHECKCNT=$DEFAULTCHECKCNT+1
    fi
    typeset -i ERRORLOGCNT=$ERRORLOGCNT
    typeset -i QUORUMCNT=$QUORUMCNT
    typeset -i MIRRORSAMEPVCNT=$MIRRORSAMEPVCNT
    typeset -i MIRRORCNTPVCNT=$MIRRORCNTPVCNT
    typeset -i MIRRORPVOFFCNT=$MIRRORPVOFFCNT
    if [ $ERRORLOGCNT != 0 ]
      then
      if [ $ERRORLOGCNT -ge $ERRORLOGLIMIT ]
        then
          ERRORLOGCNT=0
        else
          ERRORLOGCNT=$ERRORLOGCNT+1
      fi
      echo "ERRORLOGCNT $ERRORLOGCNT"  >> /tmp/Counters
    else
      echo "ERRORLOGCNT $ERRORLOGCNT"  >> /tmp/Counters
  fi
  if [ $QUORUMCNT != 0 ]
    then
      if [ $QUORUMCNT -ge $QUORUMLIMIT ]
        then
          QUORUMCNT=0
        else
          QUORUMCNT=$QUORUMCNT+1
      fi
      echo "QUORUMCNT $QUORUMCNT"  >> /tmp/Counters
    else
      echo "QUORUMCNT $QUORUMCNT"  >> /tmp/Counters
  fi
  if [ $MIRRORSAMEPVCNT != 0 ]
    then
      if [ $MIRRORSAMEPVCNT -ge $MIRRORSAMEPVLIMIT ]
        then
          MIRRORSAMEPVCNT=0
        else
          MIRRORSAMEPVCNT=$MIRRORSAMEPVCNT+1
      fi
      echo "MIRRORSAMEPVCNT $MIRRORSAMEPVCNT"  >> /tmp/Counters
    else
      echo "MIRRORSAMEPVCNT $MIRRORSAMEPVCNT"  >> /tmp/Counters
  fi
  if [ $MIRRORCNTPVCNT != 0 ]
    then
      if [ $MIRRORCNTPVCNT -ge MIRRORCNTPVLIMIT ]
        then
          MIRRORCNTPVCNT=0
        else
          MIRRORCNTPVCNT=$MIRRORCNTPVCNT+1
      fi
      echo "MIRRORCNTPVCNT $MIRRORCNTPVCNT"  >> /tmp/Counters
    else
      echo "MIRRORCNTPVCNT $MIRRORCNTPVCNT"  >> /tmp/Counters
  fi
  if [ $MIRRORPVOFFCNT != 0 ]
    then
      if [ $MIRRORPVOFFCNT -ge MIRRORPVOFFLIMIT ]
        then
```

```
             MIRRORPVOFFCNT=0
          else
            MIRRORPVOFFCNT=$MIRRORPVOFFCNT+1
        fi
        echo "MIRRORPVOFFCNT $MIRRORPVOFFCNT"  >> /tmp/Counters
      else
        echo "MIRRORPVOFFCNT $MIRRORPVOFFCNT"  >> /tmp/Counters
    fi
fi
echo "DEFAULTCHECKCNT $DEFAULTCHECKCNT"  >> /tmp/Counters

All_Good
```

A FEW DEPENDENCIES

In preparation for the implementation of this script in your environment you should ensure that references to files that this script uses are correct. Below is a brief description of these files and their purpose.

/tmp/checkitall.log

The logfile for the shell script.

/tmp/Counters

/tmp/Counters is created by the shell script the first time it's executed. *Counters* is used by the script to prevent error notifications from being sent every time the script is run. In the event an error is corrected, *Counters* should reset the corresponding counter variable. If */tmp/ Counters* is deleted, the shell script simply regenerates it, giving all varables a default value of '0'. When the value is '0', the script notifies the administrator as soon as an error is detected and increments the counter by '1'.

/tmp/greperrors

/tmp/greperrors is used to prevent the monitoring script from notifying administrators of errors that don't require immediate attention. To create */tmp/greperrors*, simply redirect the output of **errpt -t** to */tmp/ greperrors* using **errpt -t > /tmp/greperrors**. The content of */tmp/ greperrors* should look similar to that below.

```
Id       Label           Type CL Description
00530EA6 DMA_ERR         UNKN H  UNDETERMINED ERROR
```

```
0065D888 LAN0030         Temp S  SOFTWARE PROGRAM ERROR
00D2B9FE MPS_RCVRY_EXIT  TEMP H  PROBLEM RESOLVED
01003E95 RCS_ERR_MSG_7   UNKN S  ERROR ON IFCONFIG OF A SLIP INTERFACE
019125F9 LAN801E         Temp S  SOFTWARE PROGRAM ERROR
01F2D769 X25_ALERT25     PERM H  X-25 RESTART REQUEST BY X.25 ADAPTER
02288DD6 LAN805C         Perm S  SOFTWARE PROGRAM ERROR
0375DFC2 X25_ALERT9      TEMP H  X-9  FRAME TYPE W RECEIVED
038F2580 SCSI_ERR7       UNKN H  UNDETERMINED ERROR
038F3117 MPQP_DSRDRP     TEMP H  COMMUNICATION PROTOCOL ERROR
```

Edit the */tmp/greperrors* file and remove the first line (the header containing ‘Id Label Type CL Description’). Careful consideration should then be given as to which errors are not needed. If the error is listed in the */tmp/greperrors* file, then that error is **not** reported. It may be easier to leave this file blank (although the file must exist) and add error ID entries as you encounter errors that you do not wish to be notified of. The only column in the */tmp/greperrors* file that is required is the first column, *ID*.

Jarrod Brown
AIX Programmer/Administrator (USA)

Implementing DCE for AIX

IBM’s RS/6000 is one of DCE’s reference platforms. This means that native OSF DCE releases are built specifically with the RS/6000 in mind. In addition, DCE is also supported on IBM’s Scaleable PowerParallel SP2 system. While the SP2’s PSSP control program uses its own Kerberos authentication system, this does not interfere with DCE. Therefore DCE runs on SP2 nodes as it does on any other AIX system.

The main product for AIX is IBM’s DCE for AIX Version 2.2, which is based on OSF DCE Release 1.2.2.

IBM DCE FOR AIX

While DCE client services ship with the operating system itself, DCE for AIX contains the core implementation and the main components: Remote Procedure Call (RPC), Cell Directory Service, Global Directory Agent (which provides access to the global directory based on CCITT X.500/ISO), Distributed Time Service (which provides synchronized time in the distributed network environment to all computers participating in the environment), Security Service, DCE Web Administration Utilities, and DCE Threads (which are based on the POSIX 1003.4a Draft 4 standard and are mapped to native AIX threads).

The Global Directory Service (or GDS) is not provided in release 2.2. GDS can, however, exist within the same cell and be used for inter-cell communications as long as the service is provided by another product in the cell, such as DCE for Windows NT.

Other servers and functions are options that are additionally available. Among them are DCE Enhanced Distributed File System for AIX, DCE NFS to DFS Authentication Gateway for AIX, IBM DCE Manager for AIX, and IBM Getting Started with DCE for Application Developers, to mention just the most important components.

DCE SERVERS

A DCE server provides at least one DCE service, such as the Security Service. The server can be a master or a replica server. All DCE servers belong to a cell, and so they also possess all the characteristics of a DCE client system.

The different server daemons are **secd**, the security server that implements the authentication of principals and registers account information and privileges, **cdsd,** the directory server that stores and maintains object names within a cell and carries out seek requests and requests to modify and create data, and **dtsd,** the time server that provides common time in a DCE cell.

DFS is supported through EDFS (Enhanced DFS). The enhanced version adds supplemental levels of scalability and manageability, ensuring higher availability through additional functionality. The

LFS is a log-based physical file system, including data replication across EDFS servers to allow the distribution of server loads. EDFS has to be purchased separately as it is not included in the DCE for AIX base package.

The Integrated Login Facility makes administrators' lives simpler as it affords a convenient method of centrally authenticating users. The Integrated Login Facility checks the user ID and the password against the DCE registry and creates the AIX context from information in the DCE registry. It is not necessary to create local AIX accounts for users. The benefit of this should be clear as the user has a single login to both DCE and AIX. Administrators, therefore, have to maintain user accounts only in the DCE registry.

When AIX is configured for Integrated Login, the AIX login program uses the DCE Security facility to validate the user's login. From the user's point of view everything stays the same, but behind the scenes the user gets a Unix login environment and, at the same time, a DCE login environment, enabling the user to use DFS data and run DCE applications. DCE for AIX even contains programs that enable you to migrate user accounts from AIX to DCE and vice versa.

The DCE for AIX documentation is delivered on-line in hypertext format. The material has been written to conform with the Information Presentation Facility format under AIX.

INSTALLATION

You can use the **installp** command or **smit**'s install functions to install the DCE components. When installing on an RS/6000 System with AIX version 4.1.3 or higher you need the following components on the master system:

dce.cds	DCE Cell Directory Services
dce.client	DCE Client Services, Tools, and DFS Client Services
dce.sysmgmt	DCE Systems Management
dce.compat	DCE SMIT
dce.web	DCE Web Tools

dce.pthreads	DCE Threads Library
dce.security	DCE Security Services
dce.tools	Administration Tools
dce.doc	On-line documentation.

The following components are optional and may be used for enhanced cell definition:

dce.tools	DCE Application Development Tools
dce.xdsxom	DCE X.500 API Library
dce.dfs_server	DFS Base Server
dce.edfs	DFS Enhanced Server
dce.dfsnfs	NFS to DFS Authentication
dce.priv	DCE Data Encryption Library
dce.cdmf	DCE User Data Masking Encryption Facility.

In order to use DCE effectively, it is recommended that you create a new AIX JFS file system. Allow about 30 MB for the initial cell definition, which should be mounted at */var/dce*. You can find total disk requirement in the DCE for AIX manuals.

DCE clients need supplemental disk space for the DFS cache files. These are located at */var/dce/adm/fs/cache*. You can create a separate file system for this, or just enlarge */var/dce*. Monitoring the space usage of */var/dce* enables you to verify whether enough disk space is available for it.

PREPARATION FOR BUILDING CELLS

When you plan your cell's configuration, you should first gather information about its resources and business policies. Resources are people, buildings, computers, printers, and other objects that are needed to make your business operate.

Business policies describe the work performed by your company, the company's security and personnel policies, employee roles, current projects, processes used, system privileges, indispensable information, training needs and methods, etc.

You have to decide whether your company needs a single cell or multiple cells. Multiple cell environments have to be planned carefully as you may need to conduct business and communicate with other cells. Information about the nature of resources and business policies, common or different languages, cultures, currencies, operating systems, and multiple LANs or WANs spanned is important and will help you plan how many DCE objects and of what types your cell requires. You should determine a cell name and naming scheme for the DCE objects (see Figure 1, which shows an example of a cell namespace), and you should also implement a failover strategy that suits your company's needs.

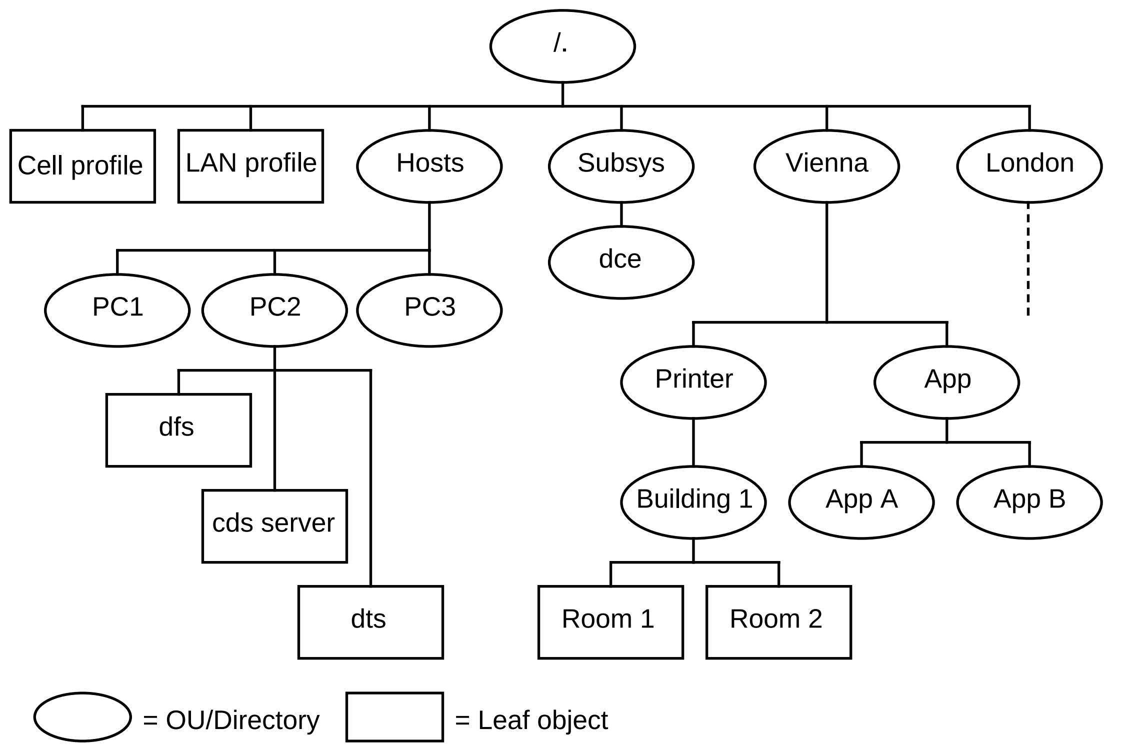

Figure 1: Example of a cell namespace

CONFIGURING A CELL

Log on as root and use the **mkdce** command to create a new DCE cell. You are prompted to enter the cell administrator's password. The related client services are created and activated automatically.

The following example creates a server with CDS, security, and DTS (time) services:

```
    >> mkdce -n aix_cell.ebner.co.at sec_srv cds_srv dts_local
```

Another way to define the cell is to use **smit**. One benefit of using **smit** is that all activities are logged in */smit.log*, another being that it’s not necessary to know the exact command syntax. This is especially useful if, for instance, you define the CDS Server and the Security Server on different machines.

Each cell needs a master Security Server. When you create the master Security Server you have to provide the name and password of the ‘principal’, who’s a kind of administrator able to perform cell administration.

If the master CDS Server is on a different system to the one on which the master Security Server resides, then you’re performing a so-called ‘split server installation’.

Every CDS server possesses a database that stores copies of directories, known as ‘replicas’. Directories are the units by which you distribute and replicate names throughout DCE namespace. Only one replica, known as the ‘master replica’, represents the original database and is writable – all the other replicas are read-only.

When you configure a CDS server as a master CDS Server you must supply the LAN profile name of your host, if the cell uses multiple LANs. If you are installing a CDS server as part of creating a new cell using CDS, the first CDS server in the cell is the master server, and each additional server is a replica server.

Once a CDS and a Security Server are installed and started, they form the nucleus of a new cell. Other systems that need to join the cell may now be configured. To join an existing DCE cell, you must provide the cell name, the hostname of the Security server, and the hostname of the CDS server. To configure an additional server, you need the name and password of a security principal who’s authorized to perform cell administration.

Sometimes it is necessary to change a cell name. An example of this is a corporate reorganization, when management may decide to reflect a new divisional structure by changing cell names, as the current ones don’t accurately reflect the organization.

The simplest solution to this problem is to create a cell ‘alias’. Cell aliases offer a high degree of flexibility and let you change the name of your cell or just add a cell name as an alias. Cell aliases can be registered with other global directory services.

You can use the *cellalias* task object within the DCE Control program to create and manage cell alias names, updating all DCE services that store the cell’s name. If you are redefining the cell name instead of simply adding an alias, you must set the new name to be the primary cell name. The *cellalias* object updates CDS, the Security Service, and the DCE hosts. The old cell name is not deleted from the services, however – the DCE Security Service and CDS retain the previous name so as to be able to respond to client requests that still use the former name.

CONFIGURING AND USING DFS

The DCE Distributed File System lets you access and store data on remote systems. This allows you to view remote filesystems as though they were local, and also extends the size of filesystems as you can integrate remote machines. DFS is built on intricate caching algorithms on client computers that allows them to store large amounts of information on data and file status. This means that the client issues fewer data requests to the file server, which, in turn, leads to reduced network traffic and server load. In addition, the file server also knows which clients have copies of data in their cache and takes the necessary steps to assure the consistency of all data.

Another advantage of DFS is that it provides a sophisticated replication system. Replication minimizes the effects of the outage of a single machine or the entire network. Frequently used filesets can be replicated to multiple servers within a cell. If a client loses access to its current server, it automatically connects to a replica server that updates their replicas on request or at pre-defined intervals. It’s important to understand exactly what ‘replicating a fileset’ means: at replication, a read-write fileset is copied from one file server to another, where the copy becomes read-only. This mechanism provides substantial benefits for read-only file access, such as storing program binaries, archives, and help files. Both systems (that is, the master server and replica) need to be configured as DFS File Server Systems and run the File Exporter process.

Replication increases performance and reliability. If the replica server is in close physical proximity to its clients, performance can be increased even more. On the other hand, you should bear in mind that replication introduces additional costs in terms of administration and load on both computers and the network as data on replica servers must be updated on a regular basis.

A DFS Server requires three different machine roles:

1 System Control Machine (SCM)

2 File Location Database Machine (FLDB)

3 Fileset Server Machine.

There is one DFS System Control Machine (the SCM), which is responsible for controlling lists of users and groups who can perform

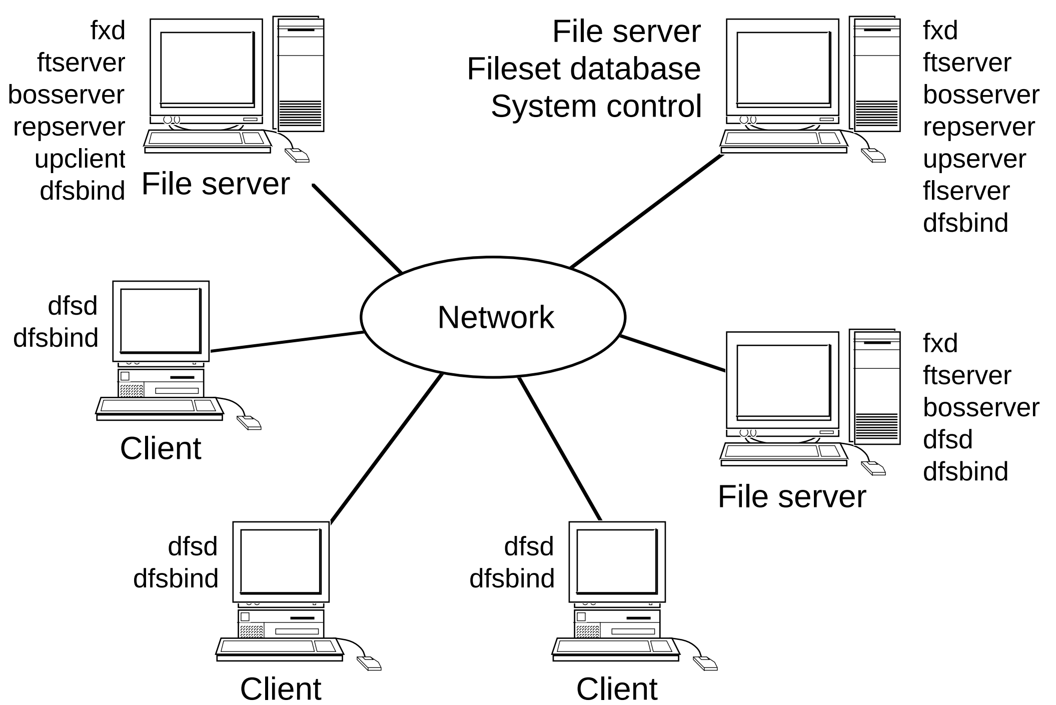

Figure 2: Example DFS configuration

administrative functions on DFS servers, collectively known as the ‘DFS domain’ or ‘administrative domain’. One DCE cell may have multiple administrative domains, but a DFS server may be a member of only one domain. Each administrative domain needs an SCM that must be the first domain system to be configured (see Figure 2, which shows an example DFS configuration).

Two important processes, *upserver* and *bosserver*, run on the SCM. *upserver* controls the distribution of administrative lists to other servers in the domain, and *bosserver* is the ‘Basic Overseer Server’ process; it’s not associated with any specific machine role, and instead supervises processes that are listed in the *BosConfig* configuration file stored in */opt/dcelocal/var/dfs*.

The *upclient* process runs on DFS Fileset Server machines; it contacts the *upserver* process running on the SCM every five minutes to synchronize its own administrative lists.

The Fileset Location Database Machine stores the Fileset Location Database (FLDB). This database is able to determine the location of a file’s server given the name of the file in DFS name space. This means that users don’t need to know where files are physically located – instead, they just get a complete, hierarchical view of the entire DFS file space.

A ‘fileset’ is a subtree of files and directories that is smaller than or equal to a logical disk partition. The Fileset Server Machine stores and exports DFS LFS filesystems or non-LFS filesystems to the DFS name space. Note that AIX’s JFS is a non-LFS filesystem. DCE for AIX supports JFS, LFS, and CD-ROM filesystems.

When DFS clients are active, they keep a local cache of files recently accessed or updated. DFS clients and DFS servers keep track of the status of all cached files using a token-passing mechanism. Clients hold tokens to files they have cached for read or update purposes. As DFS clients cache files, subsequent access to the same data in the file is very efficient, and delays are experienced only when reading files that are larger than the DFS cache.

Installing the DFS client requires the preliminary installation only of the DCE core services. You are then required to mount a separate filesystem for the DFS cache, the default size being 10 megabytes. If you want to select a different cache size, you should do so via **smit**. When the default value is convenient for you, the DFS client can be installed using the following command:

```
# mkdfs dfs_cl
```

FILE SERVERS AND FILESETS

The first step is to create file servers and configure them with DCE and (usually) the DFS or EDFS service. An option at this point is to configure systems for replication and a DFS back-up database.

Clients need to access resources on the network – in other words, they need access to DFS namespace. To make data available from file servers, you have to export the data and create filesets from a File Server using either DCE LFS, DCE DMLFS, AIX JFS, or AIX CD-ROM. Name the first fileset you create *root.dfs*.

This fileset, *root.dfs*, is the top-level fileset in DFS. It's necessary to configure *root.dfs* when you want to set up the DFS namespace on the File Server. You can configure any filesystem as the *root.dfs* except AIX CD-ROM, and you have to bear in mind that, while using an AIX JFS fileset, you cannot use DFS's fileset replication feature. For these reasons, it is recommended that you use either DCE LFS or DCE DMLFS as your *root.dfs*.

To initiate the process of exporting a DCE LFS, aggregate it from a DFS File Server if the DCE 2.2 for AIX Enhanced Distributed File System LPP has been installed. To do this, log in as a DCE user with sufficient rights to perform DFS aggregate and fileset tasks:

```
dce_login cell_administrator
```

As *root*, start **smit** using the following 'fastpath':

```
smit dfslfs
```

This article concludes in next month's issue of AIX Update.

Klaus Ebner
Curriculum Manager
IBM Austria E+T (Austria)

IBM has announced Check Point FireWall-1 Version 4.0 for AIX, providing integrated Internet and intranet access control and authentication and encryption services for secure remote access and virtual private networks. An entire network security policy can be created, monitored, and maintained from a single RS/6000 workstation. It runs on AIX 4.3.2 or 4.2.1, and is shipping now. Prices start at US$900 and go up to US$5,700 for the Enterprise Centre licence.

IBM also announced eNetwork Firewall for AIX version 3.3, now with IPSec technology for connecting branch offices together using the Internet with up to Triple DES strength encryption at each branch. It's out now, and prices start at US$2,500.

Separately, IBM announced a new AIX Bonus Pack for AIX 4.3. It's included with every new order of AIX Version 4.2 or 4.3, while existing licensees can order it separately. The Bonus Pack comes with numerous tools, including Geodesic Systems Great Circle 3.1, IBM Java MediaFramework (JMF) Version 1.1.0.1, the Kernel Group ZeroFault Dynamic Debugger Version 2.3, and a 30-day trial of Oracle8. The Bonus Pack is out now and is available at no additional charge.

For further information contact your local IBM representative.

* * *

Data General has announced CLARiiON Remote Mirroring disaster recovery, a combination of hardware, software, and services to protect against unplanned outages. Planned to be the first in a series of announcements, the Remote Mirroring is available on the FC5700 Fibre Channel RAID disk arrays as well as on the FC5500 and C3500 Series. It's to be available on AIX, Solaris, HP-UX and NT.

For further information contact:
Data General Corp, 4400 Computer Drive, Westboro, MA 01580, USA
Tel: +1 508 898 5000
Fax: +1 508 366 6591
Web: http://www.dg.com

Data General Ltd, Data General Tower, Great West Road, Brentford, Middx TW3 9AN, UK
Tel: +44 181 758 6000
Fax: +44 181 758 6758

* * *

Oracle has at last started shipping Oracle8i, first announced last September and promised before the end of 1998. The AIX version is in the first batch to be released, along with NT.

For further details contact:
Oracle Corp, PO Box 659506, Redwood Shores, CA 94065, USA
Tel: +1 650 506 7000
Fax: +1 650 506 7200
Web: http://www.oracle.com

Oracle Corp UK Ltd, The Oracle Centre, The Ring, Bracknell, Berks, RG12 1BW
Tel: +44 1344 860066
Fax: +44 1344 860222