61 AIX update

November 2000

In this issue

Published by

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

North American office

Xephon/QNA
Post Office Box 350100, Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

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

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

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/aixupdate.html* (you'll need the user-id shown on your address label to access it).

Printed in England.

Message catalogues

How many people must have browsed scripts provided by IBM and other software suppliers and wondered what on earth all those funny **dspmsg** lines mean, with their strange and unfathomable syntax? Well read on – you are about to find out!

Within AIX, message catalogues are used by both shell scripts and C programs. However, this article restricts itself to their use in scripts. If you want to know how to use them in C programs, then you'll have to get a good programming book!

IBM documentation is sadly lacking when it comes to creating and using message catalogues, and what little does exist is not easy to understand. Consequently, a lot of trial and error is required if you are trying to create a catalogue from scratch without access to either decent documentation or someone who's done it all before.

With luck, this article should allow you to create and use simple message catalogues using either numeric or symbolic identifiers (explained later). The article does not, however, cover the intricacies, subtleties, and advanced techniques of AIX message catalogues. If you want to discover these, then you must look elsewhere for inspiration (or wait for another article!).

ADVANTAGES AND DISADVANTAGES

So, why do we use message catalogues instead of using simple **echo** or **print** statements, and what are their advantages and disadvantages?

The advantages are:

- All messages are contained in catalogues that are external to a script, so that you don't have to modify the script to change the content of a particular message. Every time you edit a script, you risk adding bugs; this can be avoided with message catalogues. As we shall see later, this may not be quite the advantage it seems.
- If you're a software developer, and your software is to be used in different countries, then it is relatively simple to develop message

catalogues in different languages without having to change your scripts or C programs; this is arguably the biggest advantage of using message catalogues.

- People think you are a cool and knowledgeable shell script programmer when you use message catalogues. Little do they know!

And the disadvantages:

- **echo** and **print** are relatively easy to use. Using **dspmsg** makes your scripts much more complicated.
- It is often exceedingly difficult to determine what the wording of your message is going to be from the script.
- Developing and testing takes longer as you have to create the message catalogues in addition to the script.

MESSAGE CATALOGUE TEXT FILES

Message catalogues are created from simple text files. Once you are satisfied with the message content, there are a couple of commands that you can use to create the catalogue itself, depending on the format of the text files themselves (more on this later).

The format of the text files from which message catalogues are built follows some simple (but difficult to ascertain) rules, and it's probably easiest to use examples to illustrate the rules. Let's assume that we have a number of scripts that perform weekly **mksysb**s of our system and take daily and weekly back-ups of a number of business-critical applications. The various scripts display messages on the console, place entries in log files, and send messages to administrators via Lotus Notes when a particular events happen in a script, such as when a back-up starts or errors occur.

We have decided to use a single message catalogue that is accessed by all of our scripts. The text file used to generate the catalogue is shown below.

```
$ ----------------------------------------------------
$set 1 General
$ ----------------------------------------------------
```

```
$ Log message: Date
1 %s %s %s %s %s:

$ Log and console message: Tape drive not in available state
2 %s is not in "Available" state \n

$ Log message: Error rewinding tape
3 Error rewinding %s \n

$ Notes message: Can't make tape drive available
4 System backup could not be started \n\
%s could not be put in an "Available" state \n
.
.
.

$ -----------------------------------------------------
$set 2 mksysb backups
$ -----------------------------------------------------

$ Log message
1 Starting mksysb \n

$ Log message
2 ERROR - mksysb failed \n

$ Notes message: mksysb failed - see log
3 ERROR - mksysb failed \n\
See %s \n

$ Notes message: mksysb verification failed - see log
4 ERROR - mksysb verification failed \n\
See %s \n
.
.
.
```

COMMENT LINES

In the above example, any line that starts with a *$* symbol is a comment, provided that the *$* is immediately followed by one or more spaces or tabs. Comments are particularly useful when you have cryptic messages (that is, messages that are cryptic when not in context – naturally, no programmer would ever write cryptic error messages!) to make it immediately clear what the messages are for.

USING SETS

We have also used sets to break the file into easily distinguishable sections. A set is created when the word *set* immediately follows a *$*, with no spaces between the *$* and the 's' of 'set'. The *set number* must follow the word *set.* Set numbers don't have to be consecutive, but they must be unique. Any text that follows the number is treated as a comment.

In the above example *set 1* is used for general messages, which might be used by all scripts that access the same catalogue. We use *set 2* for messages produced by the script that runs our **mksysb**, and we can create any number of additional sets for other back-up scripts.

You don't have to have a single set in the text file. However, if you have numerous messages, or a number of different scripts use the same catalogue, then it may be sensible to split the catalogue using sets. When you make changes later, large text files that are organized in sets are easier to administer.

MESSAGES NUMBERS

Within each set we can create any number of messages, and, in the above example, each is identified by a unique number at the start of the line followed by a space or tab. Symbolic identifiers, which are discussed shortly, can also be used. Message numbers do not have to be unique within the catalogue as a whole, though they must be unique within a set. As with sets, message numbers do not have to be consecutive.

Any text following the space after the message number constitutes the first line of the message.

MESSAGE SYNTAX

The syntax of each message bears a strong similarity to that used in C, so you should have few problems in extending the simple messages shown above to produce output formatted to your own taste, if you are familiar with the C *printf* statement.

If there is a single line of text in a message and you end the line with

a *\n*, then the message is followed by a carriage return when it is printed; if you don't include a *\n*, the next message appears on the same line. For example, message *1* in set *1* produces a formatted date; when it's immediately followed by message *1* in set *2*, the resulting output is similar to:

```
Fri 8 Jan 2000 21:00: Starting mksysb
```

If a message includes more than one line, such as message *4* in set *2*, the continuation character, '\', must be used at the end of all lines except the last one (with or without a preceding *\n* – if you don't use a *\n*, you may have to put spaces before the backslash to ensure the message is readable).

Variables can be passed to messages – as shown in the above example, the location where a string variable should be inserted in a message is marked by a *%s*. You can use a *%d* to indicate where an integer is to be inserted in a message and any of the other *printf* formatting characters, though I find that *%s* suffices for most messages. You should be aware that, if you include a *%d* in a message and pass the message a character string instead of the expected integer, you'll get an error.

You should also be aware that, when you pass a string to a message and the string is framed in quotes as it contains one or more spaces, then each space acts as a string separator as far as the message catalogue is concerned. In such situations, you must have a *%s* for each space-separated group of characters within the quoted string passed to the message.

For example:

```
dspmsg . . . . . "Fri  8  Jan  2000  21:00"
```

(ignoring, for the moment, other arguments of **dspmsg**) requires five *%s* entries in the corresponding line in the message catalogue, an example being message *1* in set *1* above.

USING SYMBOLIC SET AND MESSAGE IDENTIFIERS

It's pretty easy to display messages from within your script by identifying the message using set and message numbers, though once

you've created your message catalogue you've either got to remember what the numbers are or write them down somewhere to refer to. Wouldn't it be so much simpler if you could use string identifiers, which are so much easier to remember? Well you can, but the commands that are used to generate message catalogues from such files are different from those that are used when messages are identified by set and message numbers in the source file.

The following example is taken from a message catalogue used by a notification script that handles various events that are generated by IBM's High Availability Cluster Multi Processing (HACMP) software. If certain events occur, such as a node failing within a cluster, then we want to extract a message associated with the event from the message catalogue and send it someone.

```
$ ----------------------------------------------------
$set node_events
$ ----------------------------------------------------

$ node_down
node_down %s has left its HACMP cluster \n\
The node may have crashed \n\
Resources have been taken over and are available on %s \n

$ node_up_complete
node_up_complete %s has rejoined its HACMP cluster \n

$ ----------------------------------------------------
$set network_events
$ ----------------------------------------------------

$ fail_standby
fail_standby Standby adapter with address %s has failed \n\n\
The failure may be due to: \n\
\t A resource takeover from another cluster node \n\
\t Failure of the adapter itself \n\
\t Failure of the cable from the adapter to the network \n\
\t Disconnection of the cable from the adapter to the network \n\
\t Failure of network hub to which the adapter is connected \n\
\t Interface has been brought down with the ifconfig command \n

$ join_standby
join_standby Standby adapter with address %s has rejoined \n\
the cluster \n
```

In this example, we use symbolic string identifiers for both sets and messages. Symbolic identifiers replace the set and message numbers

in the earlier example. We shall see how we can convert this text file into a message catalogue, and how it is accessed during scripts, shortly.

CREATING A MESSAGE CATALOGUE

Once you are satisfied that your text file contains messages in the right format, the messages must be converted into catalogue entries that can be used by both your scripts and the **dspmsg** command. The way in which text files are converted to catalogues depends on whether you use numeric or symbolic identifiers.

SETS AND MESSAGES USING NUMBERS

The **gencat** command is used to create catalogues of messages that are identified by set and message numbers. Its format is:

```
gencat  catalogue_file  text_source_file
```

For example:

```
gencat  backups.cat  backups.txt
```

The command creates the catalogue file, if it doesn't already exist, or overwrites it, if it does. Every time you change messages in the text file, you must run **gencat** again. You can specify any number of text files on the command line and each will modify the catalogue file as it is processed. Set and message numbers need not be unique through all text files processed by **gencat** – if there is a conflict, the last set/message overwrites a preceding one in the catalogue file.

SETS AND MESSAGES USING SYMBOLIC IDENTIFIERS

We use the **runcat** command to generate message catalogues from text files that contain symbolic identifiers. The **man** pages say that you can achieve the same result using the **mkcatdefs** command and then piping the output to **gencat**, but why use two commands when one will do?

The syntax of this command is:

```
runcat  catalogue_parameter  text_source_file
```

The text source file contains the messages, including symbolic identifiers. For example, to generate a catalogue called *backups.cat* from the source file *backups.txt*, enter:

```
runcat  backups  backups.txt
```

The command also generates a definition file whose name is constructed by adding *_msg.h* to the end of *catalogue_parameter*. In the above example, this is the *backups_msg.h* file, whose contents are similar to those of a C header file. The header-like file is used by scripts to convert symbolic set and message identifiers to numeric identifiers, and it's the numeric identifiers that are used by **dspmsg** to extract messages from the message catalogue.

In the HACMP example above, the following definition file (called *events_msg.h*) is produced:

```
#ifndef _H_EVENTS_MSG
#define _H_EVENTS_MSG
#include <limits.h>
#include <nl_types.h>
#define MF_EVENTS "events.cat"

/* The following was generated from events.msg. */

/* definitions for set node_events */
#define node_events 1

#define node_down 1
#define node_up_complete 2

/* definitions for set network_events */
#define network_events 2

#define fail_standby 1
#define join_standby 2
.
.
.
#endif
```

The *node_events* set symbolic identifier is converted to numeric set *1*, while the *network_events* set becomes set *2*. Similarly, the *node_down* message becomes message *1*, and the *node_up_complete* message becomes message *2*.

LOCATION OF MESSAGE CATALOGUES

Many message catalogues are stored in one of the default sub-directories of */usr/share/man*, principally *catx* and *manx*, though it's not mandatory that your own catalogues reside here for you to access them from your scripts. The **info** command (pre AIX 4.3) also uses message catalogues, though they are in a different location.

dspmsg automatically scans the default directories to establish whether a catalogue exists, so you don't need to specify a path name for your catalogue when you use **dspmsg** if your catalogue (and its associated *_msg.h* definition file) is located in any of the sub-directories of */usr/share/man*. However, make sure that your catalogue name doesn't conflict with an existing one. If you intend to use a directory that's specific to your application, then you must specify the full pathname of the catalogue when you use it in a script, otherwise you will not be able to extract messages.

DISPLAYING MESSAGES FROM CATALOGUES

We use the **dspmsg** command to display a message from a catalogue ('display' actually means 'retrieve'). The syntax of the command depends on whether you use sets, the location of the catalogue, and how you require default messages to be displayed in the event that a message catalogue cannot be accessed (for whatever reason). The fussier you are, the more work is involved.

The syntax of the command is:

```
dspmsg [-s set_no] cat_file msg_no 'default_msg' [arguments]
```

Points to note are:

- The set number is optional. If you omit it, the command defaults to the first set. If you use a catalogue that contains no sets, then don't use the **-s** option.
- Specify the full path name of the catalogue file if it's not a sub-directory of */usr/share/man*, otherwise just use the file's base name.
- The default message must be in quotes if it contains spaces (more on default messages later).

- You must pass all arguments that the message expects, otherwise you'll get an error.
- If **dspmsg** cannot find a message specified by a set and message number, then it displays nothing and returns no error message.
- **dspmsg** uses set and message numbers, so you must extract these numbers from the *_msg.h* definition file before you can display your message if your catalogue uses symbolic identifiers. This increases the complexity of your script, though it's relatively easy to do. Doing this also means that you don't have to remember the numbers themselves. If we take the HACMP events example above, you can easily extract the messages using code similar to the following:

```
CATFILE=/usr/local/events.cat
HDRFILE=/usr/local/events_msg.h
.
.
f_send_message()
{
   SET=$1
   EVENT=$2
   SETNO=$(grep "^#define $SET " $HDRFILE | awk '{print $3}')
   MESSNO=$(grep "^#define $EVENT " $HDRFILE | awk '{print $3}')
   MESSAGE=$(dspmsg -s $SETNO $CATFILE $MESSNO '')
.
.
}
```

When you call the *f_send_message* function, the symbolic set identifier is *$1* and the symbolic message identifier is *$2*. For example:

```
f_send_message  node_events  node_down
```

DEFAULT MESSAGES

One of the big drawbacks of using message catalogues is that you must create a default message if you want to be absolutely certain that a message is displayed even if the catalogue itself is missing.

This means that the default messages themselves must be contained either in the body of the script or in a text file that can be accessed by the script. This increases the administrative overhead and requires

you to translate both the message catalogues and the default messages, if you want your scripts to display messages in a different language.

The easy way out is to leave the default messages in the original language, or – even easier – use blank (‘’) default messages, which never require maintenance. If you generated the catalogue for use on a machine where you can be reasonably sure that the catalogue will never go missing (or at least you can easily replace it if it does), then it seems pointless and time-consuming to use default messages other than blank ones. However, developers of products that are sold to the general public probably won’t have this luxury and must assume that, out in the real world, someone is going to do something unpredictable with his or her message catalogues. Always assume the worst!

Tonto Kawalski (UK)

Directory listing

dl is a shell script that lists all the directories and files recursively, starting at a specified directory. While this facility already exists in AIX as an option of the **ls** command, this script provides custom formatting of the output. The script also demonstrates the use of recursive function calls to process subdirectories.

DL.SH

```
################################################################################
# Name     : dl.sh (directory listing)
#
# Overview : The script lists all the files and sub-directories of
#            a given top-level directory.
#
# Notes    1. The script contains the following functions:
#             - InitializeVariables
#             - GetDirectoryName
```

```
#             - ProcessExit
#             - HandleInterrupt
#             - DisplayMessage
#             - MoveCursor
#             - ProcessMainDirectory
#             - ViewReport
#             - PrintFile
#             - FormatUnderscores
#             - ProcessSubDirectory
#
#          2. The script displays an interactive message if it
#             can't switch to a directory.
##############################################################################
##############################################################################
# Name     : InitializeVariables
#
# Overview : Initializes all the variables.
##############################################################################
InitializeVariables ()
{
#
# top-level directory provided by user
#
DIR_NAME=
#
# file containing all sub-directories names in the
# $DIR_NAME directory
#
LEVEL1_SUB_DIR_FILE="/tmp/dl_$$_l1.dir"
#
# temporary files
#
TEMP_FILE_1="/tmp/dl_$$_1.tmp"
TEMP_FILE_2="/tmp/dl_$$_2.tmp"
REPORT_FILE="/tmp/dl_$$.rep"
ERROR_FILE="/tmp/dl_$$.err"
INDEX=0
#
# define escape sequences
#
ESC="\0033["
RVON="\0033[7m"                 # reverse video on
RVOFF="\0033[27m"               # reverse video off
BOLDON="\0033[1m"               # bold on
BOLDOFF="\0033[22m"             # bold off
BON="\0033[5m"                  # blinking on
BOFF="\0033[25m"                # blinking off
#
# define sleep duration
#
```

```
SLEEP_DURATION=3           # used by DisplayMessage ()
#
# messages
#
OSERROR="\${ERR_MSG}${RVOFF}"
INTERRUPT="Program interrupted\; quitting early${RVOFF}"
INVALID_ENTRY="Invalid entry${RVOFF}"
INVALID_DIR="\${DIR_NAME} is an invalid directory${RVOFF}"
CHANGE_DIR_FAILED="Cannot switch to directory \${DIR}${RVOFF}"
WORKING="Working...${RVOFF}"
PRINT_OK="Successfully submitted print job${RVOFF}"
PRINT_NOT_OK="Failed to submit print job${RVOFF}"
#
# exit codes
#
SEC=0   # success exit code
FEC=99  # failure exit code
#
# return codes
#
TRUE=0
FALSE=1
ERROR="${RVON}dl.sh:ERROR:"
INFO="${RVON}dl.sh:INFO:"
#
# define  signals
#
SIGHUP=2    ;  export SIGHUP     # when session disconnected
SIGTERM=15  ;  export SIGTERM    # kill command
SIGTSTP=18  ;  export SIGTSTP    # ctrl-z command
}
##############################################################################
# Name     : HandleInterrupt
#
# Overview : Calls ProcessExit.
##############################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit  $FEC
}
##############################################################################
# Name     : MoveCursor
#
# Input    : Y and X coordinates
#
# Returns  : None
#
# Overview : Moves the cursor to the required location (Y, X).
##############################################################################
```

```
MoveCursor  ( )
{
trap "HandleInterrupt " $SIGINT  $SIGTERM $SIGHUP
YCOR=$1
XCOR=$2
echo "${ESC}${YCOR};${XCOR}H"
}
################################################################################
# Name     : DisplayMessage
#
# Overview : The function displays message
#
# Input    : 1. Message type (E = Error, I = Information)
#            2. Error code defined in DefineMessages ().
################################################################################
DisplayMessage ( )
{
MESSAGE_TYPE=$1
MESSAGE_TEXT=`eval echo $2`
MoveCursor 24 1
if [ "${MESSAGE_TYPE}" = "E" ]
then
    echo "`eval echo ${ERROR}`${MESSAGE_TEXT}\c"
else
    echo "`eval echo ${INFO}`${MESSAGE_TEXT}\c"
fi
sleep ${SLEEP_DURATION}
return ${TRUE}
}
################################################################################
# Name     : ProcessExit
#
# Overview : Makes a graceful exit.
#
# Input    : Exit code
################################################################################
ProcessExit ()
{
EXIT_CODE="$1"
#
# remove all temporary files
#
rm /tmp/dl_*
exit $EXIT_CODE
}
################################################################################
# Name     : FormatUnderscores
#
# Overview : The function assigns the right number of underscore
#            characters (=) to the variable UNDERSCORE, to be used
```



```
#            in conjunction with a header.
#
# Input    : Line containing the header
################################################################################
FormatUnderscores ()
{
#
# assign parameter
#
LINE="$1"
#
# initialize UNDERSCORE
#
UNDERSCORE=
#
# initialize index
#
IND=1
#
# get no of characters in $LINE
#
NO_CHARS=`echo "$LINE" | wc -c`
#
# subtract the carriage return
#
NO_CHARS=`expr $NO_CHARS - 1`
while [ "$IND" -le "$NO_CHARS" ]
do
    UNDERSCORE="${UNDERSCORE}="
    IND=`expr $IND + 1`
done
}
################################################################################
# Name        : PrintFile
#
# Description : Prints the named file
#
# Input       : File name to be printed
################################################################################
PrintFile ( )
{
FILE_TO_BE_PRINTED=$1
#
# print file
#
while true
do
    clear
    echo "Do you wish to print the output file (Y/N)?:\c"
    read REPLY
```

```
    case $REPLY in
        n|N ) return $TRUE ;;
        y|Y ) break ;;
          * ) : ;;
    esac
done
#
# get printer name
#
while true
do
    clear
    echo "Enter printer name for lp command (q to quit):\c"
    read PRINTER
    case $PRINTER in
         "" ) : ;;
        q|Q ) break ;;
          * ) lp -d$PRINTER ${FILE_TO_BE_PRINTED} > /dev/null 2>&1 ;
              if [ $? -eq 0 ]
              then
                  DisplayMessage I "${PRINT_OK}" ;
                  break ;
              else
                  DisplayMessage E "${PRINT_NOT_OK}" ;
              fi ;;
    esac
done
}
##############################################################################
# Name     : GetDirectoryName
#
# Overview : Obtains a directory name from the user.
##############################################################################
GetDirectoryName ()
{
while  true
do
    clear
    echo "Enter the directory's full name (q to quit):\c"
    read DIR_NAME
    case $DIR_NAME in
         "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
        q|Q ) return $FALSE ;;
          * ) if [ -d ${DIR_NAME} ]
              then
                  return $TRUE;
              else
                  DisplayMessage E "${INVALID_DIR}";
              fi ;;
```

```
    esac
done
}
#############################################################################
# Name     : ProcessMainDirectory
#
# Overview : Processes files and directories in the top-level
#            directory (the directory named by the user).
#############################################################################
ProcessMainDirectory ()
{
DisplayMessage I "${WORKING}"
#
# initialize report file
#
HEADER="File Listing for $DIR_NAME"
FormatUnderscores "${HEADER}"
echo "${HEADER}" > ${REPORT_FILE}
echo "${UNDERSCORE}" >> ${REPORT_FILE}
#
# switch to the directory
#
cd $DIR_NAME > ${ERROR_FILE} 2>&1
if [ $? -ne 0 ]
then
    DIR="${DIR_NAME}"
    DisplayMessage E "${CHANGE_DIR_FAILED}"
    ERR_MSG=`cat ${ERROR_FILE}`
    DisplayMessage E "${OSERROR}"
    return $FALSE
fi
ls -1 > ${TEMP_FILE_1}
cat ${TEMP_FILE_1} | while read FILE_NAME
do
    if [ -d ${FILE_NAME} ]
    then
        #
        # file is a directory
        #
        echo "${DIR_NAME}/$FILE_NAME" >> $LEVEL1_SUB_DIR_FILE
        echo "Directory -----> $FILE_NAME  " >> ${REPORT_FILE}
    else
        echo "File ----------> $FILE_NAME" >> ${REPORT_FILE}
    fi
done
ProcessSubDirectory "${LEVEL1_SUB_DIR_FILE}"
return $TRUE
}
#############################################################################
```

```
# Name     : ProcessMainDirectory
#
# Overview : Process files and directories in sub-directories
#            recursively.
#
# Notes    1. This function calls itself.
#
#          2. The variable SUB_DIR_FILE stores the name of a file
#             that contains a list of directories to be processed.
#
#          3. The variable DIR_TO_BE_PROCESSED stores the name of a
#             file that contains a list of directories found in the
#             directory being processed. This is part of the
#             processing of all directories in the file pointed to by
#             $SUB_DIR_FILE.
################################################################################
ProcessSubDirectory  ()
{
SUB_DIR_FILE="$1"
INDEX=`expr $INDEX + 1`
DIR_TO_BE_PROCESSED="/tmp/dl_$$_$INDEX.dir"
cat ${SUB_DIR_FILE} | while read DIR
do
    HEADER="File listing of $DIR"
    FormatUnderscores "${HEADER}"
    echo "\n${HEADER}" >> ${REPORT_FILE}
    echo "${UNDERSCORE}" >> ${REPORT_FILE}
    #
    # switch to the directory
    #
    cd $DIR > ${ERROR_FILE} 2>&1
    if [ $? -ne 0 ]
    then
        DisplayMessage E "${CHANGE_DIR_FAILED}"
        ERR_MSG=`cat ${ERROR_FILE}`
        DisplayMessage E "${OSERROR}"
        echo "Cannot switch to this directory" >> ${REPORT_FILE}
        clear
        continue
    fi
    ls -1 > ${TEMP_FILE_1}
    cat ${TEMP_FILE_1} | while read FILE_NAME
    do
        if [ -d ${FILE_NAME} ]
        then
        #
        # file is a directory
        #
            echo "$DIR/$FILE_NAME" >> ${DIR_TO_BE_PROCESSED}
```

```
            echo "Directory -----> $FILE_NAME  " >> ${REPORT_FILE}
        else
            echo "File ----------> $FILE_NAME" >> ${REPORT_FILE}
        fi
    done
    if [ -s ${DIR_TO_BE_PROCESSED} ]
    then
        ProcessSubDirectory ${DIR_TO_BE_PROCESSED}
    fi
done
return $TRUE
}
##############################################################################
# Name     : ViewReport
#
# Overview : Allows the user to view and print the listing.
#
# Notes    1. This function calls the following function:
#             - PrintFile
##############################################################################
ViewReport ()
{
view ${REPORT_FILE}
PrintFile "${REPORT_FILE}"
}
##############################################################################
# Name     : main
#
# Overview : Implements the processing structure.
#
# Notes    1. The function calls the following functions:
#             - InitializeVariables
#             - GetDirectoryName
#             - ProcessMainDirectory
##############################################################################
main ()
{
InitializeVariables
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP
if ! GetDirectoryName
then
    ProcessExit $FEC
fi
if ! ProcessMainDirectory
then
    ProcessExit $FEC
fi
ViewReport
ProcessExit
```

```
}
#
# invoke main
#
main
```

SAMPLE OUTPUT

```
File listing of
=======================
Directory -----> anonymou
Directory -----> ecatmgr
File listing of /home/anonymou
================================
File listing of /home/ecatmgr
================================
File ----------> a.sql
Directory -----> appl
File ----------> b.sql
File ----------> c.sql
File ----------> nohup.out
File ----------> pman.sh
File ----------> q1.sql
Directory -----> sh
File ----------> smit.log
File ----------> smit.script
Directory -----> utils
File ----------> vim.log
File listing of /home/ecatmgr/appl
===================================
Directory -----> dbalert
Directory -----> ecatwebd
File listing of /home/ecatmgr/appl/dbalert
============================================
File listing of /home/ecatmgr/appl/ecatwebd
============================================
Directory -----> err
Directory -----> log
Directory -----> sh
Directory -----> sql
Directory -----> temp
File listing of /home/ecatmgr/appl/ecatwebd/err
================================================
File ----------> reftabs.err
File listing of /home/ecatmgr/appl/ecatwebd/log
================================================
File ----------> reftabs_03032000.log
File ----------> reftabs_06032000.log
```

```
File listing of /home/ecatmgr/appl/ecatwebd/sh
================================================
File ----------> ECATWEBORAENV.sh
File ----------> reftabs.sh
File listing of /home/ecatmgr/appl/ecatwebd/sql
================================================
File ----------> reftabs.sql
File listing of /home/ecatmgr/appl/ecatwebd/temp
================================================
File ----------> reftabs_18160.tmp
File ----------> reftabs_18372.tmp
File ----------> reftabs_18920.tmp
File ----------> reftabs_18984.tmp
File ----------> reftabs_18986.tmp
File ----------> reftabs_19730.tmp
File ----------> reftabs_21276.tmp
File ----------> reftabs_22446.tmp
File ----------> reftabs_22714.tmp
File ----------> reftabs_23480.tmp
File ----------> reftabs_23996.tmp
File ----------> reftabs_6756.tmp
File listing of /home/ecatmgr/sh
================================
File ----------> bd.sh
File ----------> cmp.sh
Directory -----> d1
File ----------> dl.sh
File ----------> emp.sh
File ----------> hi
File ----------> testpage
File ----------> typescript
File listing of /home/ecatmgr/sh/d1
====================================
Directory -----> d2
File listing of /home/ecatmgr/sh/d1/d2
========================================
Cannot switch to this directory
File listing of /home/ecatmgr/utils
====================================
File ----------> dur.sh
File ----------> mur.sh
File ----------> rsad.sh
File ----------> sp
File ----------> sqlnet.log
```

Arif Zaman
System Administrator/DBA
High-Tech Software (UK)

Maintain network details

INTRODUCTION

Maintain network details (**mnd**) is a shell script that helps with the administration of many aspects of TCP/IP. The script offers much functionality that is not available through **smit** and, therefore, supplements **smit**. While the script is fairly long (it'll be published in two parts in this and next month's issue of *AIX Update*), much of it is taken up by an embedded glossary file on tuneable TCP/IP parameters.

Note the use of the continuation character, '➤', in the code below to indicate a formatting line break that's not present in either the original source code or the code that can be downloaded from Xephon's Web site (*http://www.xephon.com/aixupdate.html*).

MND.SH

```
#! /bin/ksh
################################################################################
# Name     : mnd.sh (manage network details)
#
# Overview : The script manages aspects of networking that are not
#            available through smit.
################################################################################
################################################################################
# Name     : InitializeVariables
#
# Overview : Initializes all variables.
################################################################################
InitializeVariables()
{
# define locations
TEMP_FILE="/tmp/mnd_$$.tmp"
TEMP_FILE_1="/tmp/mnd_$$_1.tmp"
TEMP_FILE_2="/tmp/mnd_$$_2.tmp"
ERROR_FILE="/tmp/mnd_$$.err"
REPORT_FILE="/tmp/mnd_$$.rep"
# escape  sequences
ESC="\0033["
RVON="\0033[7m"            # reverse video on
RVOFF="\0033[27m"          # reverse video off
BOLDON="\0033[1m"          # bold on
```

```
BOLDOFF="\0033[22m"        # bold off
BON="\0033[5m"             # blinking on
BOFF="\0033[25m"           # blinking off
# define menu title
MND     = "${RVON}Manage Network Details${RVOFF}"
MNS     = "${RVON}Manage Network SubSystems${RVOFF}"
FCONFIG = "${RVON}Configuration File Menu${RVOFF}"
TCPIP   = "${RVON}Manage TCP/IP Details${RVOFF}"
# define exit codes
SEC=0
FEC=1
TRUE=0
FALSE=1
SLEEP_DURATION=4                          # seconds for sleep command
ERROR="${RVON}${BON}mnd.sh:ERROR:${BOFF}"
INFO="${RVON}mnd.sh:INFO: "
# messages
WORKING                 = "Working... \(do not acknowledge\)${RVOFF}"
INTERRUPT               = "Program interrupted: quitting${RVOFF}"
INVALID_REPLY           = "Invalid entry ${RVOFF}"
INVALID_USER            = "\${USER} is an invalid user${RVOFF}"
FILE_NOT_FOUND          = "File \${P_FNAME} does not exist${RVOFF}"
NEW_FILE                = "Initializing new configuration file
                          ➤  \${P_FNAME}${RVOFF}"
INVALID_NO              = "\${NO}, is an invalid network option
                          ➤  ${RVOFF}"
COMMAND_FAILURE         = "Failed to execute command \${COMMAND}
                          ➤  ${RVOFF}"
COMMAND_SUCCEEDED       = "Successfully executed command \${COMMAND}
                          ➤  ${RVOFF}"
OS_ERROR                = "\${OS_ERM}${RVOFF}"
NO_ROUTE                = "Failed to record route for \${HOST}
                          ➤  ${RVOFF}"
PACKET_LOSS             = "Failed to record route for \${HOST} as a
                          ➤  result of packet loss${RVOFF}"
PRINT_OK                = "Successfully submitted the print
                          ➤  job${RVOFF}"
PRINT_NOT_OK            = "Failed to submit the print job${RVOFF}"
DEFAULT_EXISTS          = "A default message already exists${RVOFF}"
INVALID_ENTRY           = "Invalid entry ${RVOFF}"
NOT_NUMERIC             = "Value must be numeric${RVOFF}"
FILE_NOT_INITIALIZED    = "Failed to initialize the file
                          ➤  \${FILE_NAME}${RVOFF}"
FILE_INITIALIZED        = "Successfully initialized the file
                          ➤  \${FILE_NAME}${RVOFF}"
OSERROR                 = "\${OSEM}${RVOFF}"
ROOT_USER               = "The script must be executed from the root
                          ➤  account${RVOFF}"
DEFAULT_GATEWAY_CHANGED = "Successfully changed default gateway to
```

```
                              ➤  \${GATEWAY_IP}${RVOFF}"
# define signals
SIGHUP=1    ;  export SIGHUP     # Session disconnected
SIGINT=2    ;  export SIGINT     # Ctrl-c
SIGTERM=15  ;  export SIGTERM    # Kill command
SIGTSTP=18  ;  export SIGTSTP    # Ctrl-z
}
##############################################################################
# Name     : HandleInterrupt
#
# Overview : Calls ProcessExit.
##############################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit $FEC
}
##############################################################################
# Name     : MoveCursor
#
# Input    : Y and X coordinates
#
# Overview : Moves the cursor to the required location (Y, X).
##############################################################################
MoveCursor ()
{
YCOR=$1
XCOR=$2
echo "${ESC}${YCOR};${XCOR}H"
}
##############################################################################
# Name     : DisplayMessage
#
# Overview : Displays message
#
# Input    : 1. Message type (E = Error, I = Information)
#            2. Error Code as defined in DefineMessages ().
#            3. Acknowledge message flag (Y = yes, N = no)
##############################################################################
DisplayMessage ()
{
MESSAGE_TYPE=$1
MESSAGE_TEXT=`eval echo $2`
ACKNOWLEDGE_FLAG="$3"
if [ "${ACKNOWLEDGE_FLAG}" = "" ]
then
    ACKNOWLEDGE_FLAG="Y"
fi
MoveCursor 24 1
```

```
if [ "${MESSAGE_TYPE}" = "E" ]
then
    echo "`eval echo ${ERROR}`${MESSAGE_TEXT}\c"
else
    echo "`eval echo ${INFO}`${MESSAGE_TEXT}\c"
fi
# examine message acknowledge flag
if [ "${ACKNOWLEDGE_FLAG}" = "Y" ]
then
    read DUMMY
else
    sleep ${SLEEP_DURATION}
fi
return ${TRUE}
}
################################################################################
# Name     : DisplayListOfValues
#
# Overview : Displays list of values for argument passed.
#
# Input    : String
#            NO   (network options)
#            HOST (host definitions)
#            USRR (users)
################################################################################
DisplayListOfValues ()
{
PARAM="$1"
if [ "${PARAM}" = "NO" ]
then
    echo "         List of network options" > ${TEMP_FILE}
    echo "         ======================== " >> ${TEMP_FILE}
    echo "         Select value by deleting line and saving file\n" \
        >> ${TEMP_FILE}
    no -a | cut -d'=' -f1 >> ${TEMP_FILE}
    cp ${TEMP_FILE}  ${TEMP_FILE_1}
    view ${TEMP_FILE}
    SELECTED_VALUE=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 |\
        awk {'print $2'}`
elif [ "${PARAM}" = "HOST" ]
then
    echo "         List of host definitions" > ${TEMP_FILE}
    echo "         ========================" >> ${TEMP_FILE}
    echo "         Select value by deleting line and saving file\n" \
        >> ${TEMP_FILE}
    cat /etc/hosts >> ${TEMP_FILE}
    cp ${TEMP_FILE} ${TEMP_FILE_1}
    view ${TEMP_FILE}
    SELECTED_VALUE=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | \
```

```
        awk {'print $3'}`
elif [ "${PARAM}" = "SUBSYSTEM" ]
then
    echo "         List of subsystems for TCP/IP" > ${TEMP_FILE}
    echo "         ============================" >> ${TEMP_FILE}
    echo "         Select value by deleting line and saving file\n" \
        >> ${TEMP_FILE}
    lssrc -g tcpip | awk {'print $1'} > ${TEMP_FILE_1}
    sed 1d ${TEMP_FILE_1} > ${TEMP_FILE_2}
    cat ${TEMP_FILE_2} >> ${TEMP_FILE}
    cp ${TEMP_FILE} ${TEMP_FILE_1}
    view ${TEMP_FILE}
    SELECTED_VALUE=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | \
        awk {'print $2'}`
elif [ "${PARAM}" = "USER" ]
then
    echo "         List of Users" > ${TEMP_FILE}
    echo "         =============" >> ${TEMP_FILE}
    echo "         Select value by deleting line and saving file\n" \
        >> ${TEMP_FILE}
    cat /etc/passwd | awk -F ':' {'printf("%-20s %-59s\n",$1,$6)'} \
        >> ${TEMP_FILE}
    cp ${TEMP_FILE} ${TEMP_FILE_1}
    view ${TEMP_FILE_1}
    SELECTED_VALUE=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | \
        awk {'print $2'}`
    HOME_DIR=`diff ${TEMP_FILE} ${TEMP_FILE_1} | tail -1 | awk \
        {'print $3'}`
fi
}
##############################################################################
# Name     : FormatUnderscores
#
# Overview : Assigns the appropriate number of underscores (=) to the
#            variable UNDERSCORE for use in formatting a header.
#
# Input    : A line containing the header
##############################################################################
FormatUnderscores ()
{
LINE="$1"
UNDERSCORE=
IND=1
NO_CHARS=`echo "$LINE" | wc -c`
NO_CHARS=`expr $NO_CHARS - 1`
while [ "$IND" -le "$NO_CHARS" ]
do
    UNDERSCORE="${UNDERSCORE}="
    IND=`expr $IND + 1`
```

```
done
}
################################################################################
# Name     : PrintFile
#
# Overview : Prints the named file.
#
# Input    : Name of file to be printed.
################################################################################
PrintFile ()
{
FILE_TO_BE_PRINTED=$1
while true
do
    clear
    echo "Do you wish to print the output file (Y/N)?:\c"
    read REPLY
    case $REPLY in
         n|N ) return $TRUE ;;
         y|Y ) break ;;
         *   ) : ;;
    esac
done
while true
do
    clear
    echo "Enter printer name for lp command (q to quit):\c"
    read PRINTER
    case $PRINTER in
         "" ) : ;;
         q|Q) break ;;
          * ) lp -d$PRINTER ${FILE_TO_BE_PRINTED} > /dev/null 2>&1 ;
              if [ $? -eq 0 ]
              then
                  DisplayMessage I "${PRINT_OK}" ;
                  break ;
              else
                  DisplayMessage E "${PRINT_NOT_OK}" ;
              fi ;;
    esac
done
}
################################################################################
# Name     : DisplayMNDMenu
#
# Overview : Displays the main menu.
################################################################################
DisplayMNDMenu ()
{
```

```
while true
do
clear
echo "    ##################################################"
echo "    #       $MND                #"
echo "    #                                                #"
echo "    #   5. Overview of ISO 7 layer model             #"
echo "    #  10. Display specific network option value     #"
echo "    #  15. Display all network option values         #"
echo "    #  20. Set specific network option               #"
echo "    #  25. Manage network subsystems                 #"
echo "    #  30. Manage configuration files                #"
echo "    #                                                #"
echo "    #  99. Exit                                      #"
echo "    ##################################################"
echo "             Enter option --->\c"
read OPTION
ProcessOption "MND"
done
}
#############################################################################
# Name     : DisplayMNSMenu
#
# Overview : Displays the network subsystem menu.
#############################################################################
DisplayMNSMenu ()
{
while true
do
clear
echo "    ##################################################"
echo "    #       $MNS             #"
echo "    #                                                #"
echo "    #   5. Display network subsystem groups          #"
echo "    #  10. Manage TCP/IP subsystem                   #"
echo "    #  90. Main menu                                 #"
echo "    #  99. Exit                                      #"
echo "    ##################################################"
echo "             Enter Option --->\c"
read OPTION
ProcessOption  "MNS"
done
}
#############################################################################
# Name     : DisplayNetworkSubsystemGroups
#
# Overview : Displays network subsystem group details.
#############################################################################
DisplayNetworkSubsystemGroups ()
```

```
{
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
DisplayMessage I "${WORKING}" "N"
HEADER="Report on network subsystem groups on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "   $HEADER"     > ${REPORT_FILE}
echo "   $UNDERSCORE" >> ${REPORT_FILE}
INDEX=1
NETGROUP[$INDEX]="X"
lssrc -a | awk {'print $2'} > ${TEMP_FILE}
cat ${TEMP_FILE} | while read LINE
do
    if [ "${LINE}" = "Group" ]
    then
        continue
    fi
    IND=1
    DUPLICATE=N
    while [ "${NETGROUP[$IND]}" != "" ]
    do
        if [ "${LINE}" = "${NETGROUP[$IND]}" ]
        then
            DUPLICATE = Y
            break
        else
            IND=`expr $IND + 1`
        fi
    done
    if [ "${DUPLICATE}" = "N" ]
    then
        NETGROUP[$INDEX]="${LINE}"
        INDEX=`expr $INDEX + 1`
    fi
done
INDEX=1
while [ "${NETGROUP[$INDEX]}" != "" ]
do
    HEADER="\nGroup = ${NETGROUP[$INDEX]}"
    FormatUnderscores "${HEADER}"
    echo "$HEADER"     >> ${REPORT_FILE}
    echo "$UNDERSCORE" >> ${REPORT_FILE}
    GROUP="${NETGROUP[$INDEX]}"
    lssrc -g ${GROUP} | egrep -v "Subsystem|Group|PID|Status" | \
        awk {'printf("%-10s\n",$1)'} >>${REPORT_FILE}
    INDEX=`expr $INDEX + 1`
done
view ${REPORT_FILE}
PrintFile ${REPORT_FILE}
}
```

```
#############################################################################
# Name     : DisplayTCPIPMenu
#
# Overview : The function displays the TCP/IP menu.
#############################################################################
DisplayTCPIPMenu ()
{
while true
do
clear
echo "    ##################################################"
echo "    #       $TCPIP                   #"
echo "    #                                                #"
echo "    #   5. Show All TCP/IP connections               #"
echo "    #  10. Display routing table contents            #"
echo "    #  15. Show route for a specific remote host     #"
echo "    #  20. Show all ports in use                     #"
echo "    #  25. Display notes on IP addresses             #"
echo "    #  30. Display status of a specific subsystem    #"
echo "    #  90. Main menu                                 #"
echo "    #                                                #"
echo "    #  99. Exit                                      #"
echo "    ##################################################"
echo "             Enter option --->\c"
read OPTION
ProcessOption  "TCPIP"
done
}
#############################################################################
# Name     : DisplayFileConfiguarationMenu
#
# Overview : Displays the file configuration menu.
#############################################################################
DisplayFileConfigurationMenu ()
{
while true
do
clear
echo "    ##################################################"
echo "    #       $FCONFIG                 #"
echo "    #                                                #"
echo "    #   5. View/Modify /etc/hosts       file       #"
echo "    #  10. View/Modify /etc/services    file       #"
echo "    #  15. View/Modify /etc/networks    file       #"
echo "    #  20. View/Modify /etc/gateways    file       #"
echo "    #  25. View/Modify /etc/inetd.conf  file       #"
echo "    #  30. View/Modify /etc/gated.conf  file       #"
echo "    #  35. View/Modify /etc/resolv.conf file       #"
echo "    #  40. View/Modify /etc/protocols   file       #"
```

```
echo "    #  45. View/Modify /etc/rc.tcpip    file      #"
echo "    #  50. View/Modify /etc/rc.net      file      #"
echo "    #  55. View/Modify /etc/hosts.equiv file      #"
echo "    #  60. View/Modify \$HOME/.rhost     file      #"
echo "    #  90. Main menu                               #"
echo "    #                                              #"
echo "    #  99. Exit                                    #"
echo "    ################################################"
echo "             Enter option --->\c"
read OPTION
ProcessOption "FCONFIG"
done
}
##############################################################################
# Name     : ValidateUser
#
# Overview : Validates a user.
#
# Input    : Username
#
# Returns  : $TRUE or $FALSE
##############################################################################
ValidateUser ()
{
P_USER="$1"
cat /etc/passwd | awk -F ':' {'printf("%-20s %-59s\n",$1,$6)'} \
    >> ${TEMP_FILE}
if grep "${P_USER}" ${TEMP_FILE} > ${TEMP_FILE_1}
then
    read USER HOME_DIR < ${TEMP_FILE_1}
    return $TRUE
else
    return $FALSE
fi
}
##############################################################################
# Name     : ViewEditRemoteHostFile
#
# Overview : Allows the user to view or edit a remote host
#            file.
##############################################################################
ViewEditRemoteHostFile ()
{
while true
do
    clear
    echo "Enter username for .rhost file (l for list):\c"
    read USER
    case $USER in
```

```
          "" ) DisplayMessage E "${INVALID_REPLY}" ;;
         l|L ) DisplayListOfValues "USER" ;
               USER="${SELECTED_VALUE}" ;
               if [ "${USER}" != "" ]
               then
                   break ;
               fi ;;
           * ) if ! ValidateUser "${USER}"
               then
                   DisplayMessage E "${INVALID_USER}" ;
               else
                   break ;
               fi ;;
    esac
done
FILE_NAME="${HOME_DIR}/.rhosts"
while true
do
    clear
    echo "Do you wish to view (V) or edit (E) file, "${FILE_NAME}":\c"
    read REPLY
    case $REPLY in
         v|V) if [ ! -f "${FILE_NAME}" ]
              then
                  P_FNAME="${FILE_NAME}" ;
                  DisplayMessage E "${FILE_NOT_FOUND}" ;
                  return $FALSE ;
              else
                  cp ${FILE_NAME} ${TEMP_FILE} ;
                  view ${TEMP_FILE} ;
                  return $TRUE ;
              fi ;;
         e|E) if [ ! -f "${FILE_NAME}" ]
              then
                   P_FNAME="${FILE_NAME}" ;
                   DisplayMessage I "${FILE_NOT_FOUND}" ;
                   DisplayMessage I "${NEW_FILE}" ;
                   > ${TEMP_FILE} ;
                   vi ${TEMP_FILE} ;
              else
                   cp ${FILE_NAME}  ${TEMP_FILE} ;
                   chmod u+w  ${TEMP_FILE} ;
                   vi ${TEMP_FILE} ;
              fi ;
              break ;;
    esac
done
while true
do
```

```
    clear
    echo "Do you wish to save edited file ${FILE_NAME} (Y or N):\c"
    read REPLY
    case $REPLY in
         y|Y) ( cp ${TEMP_FILE} ${FILE_NAME} > ${TEMP_FILE_2} 2>&1 ) ;
              if [ $? -ne 0 ]
              then
                  OS_ERM=`cat ${TEMP_FILE_2}` ;
                  DisplayMessage E "${OS_ERROR}" ;
                  return $FALSE ;
              fi ;;
         n|N) return $TRUE ;;
           *) DisplayMessage E "${INVALID_REPLY}" ;;
    esac
done
}
#############################################################################
# Name     : ViewEditConfigurationFile
#
# Overview : Allows the user to view or edit a named configuration
#            file.
#
# Input    : Configuration file name.
#############################################################################
ViewEditConfigurationFile ()
{
P_FNAME="$1"
while true
do
    clear
    echo "Do you wish to view (V) or edit (E) configuration file
         ➤  ${P_FNAME}:\c"
    read REPLY
    case $REPLY in
         v|V) if [ ! -f "${P_FNAME}" ]
              then
                  DisplayMessage E "${FILE_NOT_FOUND}" ;
                  return $FALSE ;
              else
                  cp ${P_FNAME} ${TEMP_FILE} ;
                  view ${TEMP_FILE} ;
                  return $TRUE ;
              fi ;;
         e|E) if [ ! -f "${P_FNAME}" ]
              then
                  DisplayMessage I "${FILE_NOT_FOUND}" ;
                  DisplayMessage I "${NEW_FILE}" ;
                  > ${TEMP_FILE} ;
                  vi  ${TEMP_FILE} ;
```

```
                else
                    cp ${P_FNAME} ${TEMP_FILE} ;
                    chmod u+w ${TEMP_FILE} ;
                    vi ${TEMP_FILE} ;
                fi ;
                break ;;
            *) DisplayMessage E "${INVALID_REPLY}" ;;
    esac
done
while true
do
    clear
    echo "Do you wish to save the edited configuration file
         ➤  ${P_FNAME}(Y or N):\c"
    read REPLY
    case $REPLY in
         y|Y) ( cp ${TEMP_FILE} ${P_FNAME} > ${TEMP_FILE_2} 2>&1 ) ;
              if [ $? -ne 0 ]
              then
                  OS_ERM=`cat ${TEMP_FILE_2}` ;
                  DisplayMessage E "${OS_ERROR}" ;
                  return $FALSE ;
              fi ;;
         n|N) return $TRUE ;;
           *) DisplayMessage E "${INVALID_REPLY}" ;;
    esac
done
}
##############################################################################
# Name     : GetServiceName
#
# Overview : Retrieves the service name for a specific port from the
#            /etc/services table.
#
# Input    : Port Number
#
# Returns  : Service
##############################################################################
GetServiceName ()
{
P_PORT="$1"
grep "${P_PORT}/tcp" /etc/services > ${TEMP_FILE_1}
S_INDEX=0
cat ${TEMP_FILE_1} | while read LINE
do
    COMMENT=`echo "${LINE} | cut -c1,1`
    if [ "${COMMENT}" = "#" ]
    then
        continue
```

```
    else
        SERVICE[$S_INDEX]=`echo "${LINE}" | awk {'print $1'}`
        S_INDEX=`expr ${S_INDEX} + 1`
    fi
done
if [ "${SERVICE[0]}" = "" ]
then
    SERVICE[0]="UNKNOWN"
fi
}
##############################################################################
# Name     : ValidClientAddress
#
# Overview : Establishes whether a given machine address is valid.
#
# Input    : String containing client address
#
# Returns  : $TRUE or $FALSE
#
# Notes    : 1. The address should be in following format:
#               <digits>.<digits>.<digits>.<digits>.<digits>
##############################################################################
ValidClientAddress ()
{
P_ADDRESS="$1"
WORD1=""
WORD2=""
WORD3=""
WORD4=""
WORD5=""
WORD1=`echo "${P_ADDRESS}" | cut -d'.' -f1`
WORD2=`echo "${P_ADDRESS}" | cut -d'.' -f2`
WORD3=`echo "${P_ADDRESS}" | cut -d'.' -f3`
WORD4=`echo "${P_ADDRESS}" | cut -d'.' -f4`
WORD5=`echo "${P_ADDRESS}" | cut -d'.' -f5`
if [ "${WORD1}" != "" -a "${WORD2}" != "" -a  \
     "${WORD3}" != "" -a "${WORD4}" != "" -a "${WORD5}" != "" ]
then
    return $TRUE
else
    return $FALSE
fi
}
##############################################################################
# Name     : DisplayAllTCPIPConnections
#
# Overview : Displays all TCP/IP client connections to the server.
#
# Notes    : 1. The client address is validated (for example, the
```

```
#               address *.* is displayed as UNKNOWN).
#
#            2. If a port is associated with more than one service,
#               only the first entry from /etc/services file is
#               displayed.
################################################################################
DisplayAllTCPIPConnections ()
{
DisplayMessage I "${WORKING}" "N"
NO_HEADER_LINES=4
NO_LINES=0
NO_CONNECTIONS=0
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Report on all active TCP/IP connections to server on
       ➤  ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "   $HEADER"     >  ${REPORT_FILE}
echo "   $UNDERSCORE" >> ${REPORT_FILE}
echo "Local_Port Service  Foreign_Address  State" | \
    awk {'printf("%-12s%-20s%-25s%-20s\n\n",$1,$2,$3,$4)'} \
    >> ${REPORT_FILE}
netstat -n > ${TEMP_FILE_1}
cat ${TEMP_FILE_1} | grep "tcp" > ${TEMP_FILE}
sed s/*.*/UNKNOWN/ ${TEMP_FILE} > ${TEMP_FILE_1}
INDEX=0
cat ${TEMP_FILE_1} | while read LINE
do
    PORT=`echo ${LINE} | awk {'print $4'} | cut -d'.' -f5`
    CLIENT_ADDRESS=`echo ${LINE} | awk {'print $5'}`
    STATE=`echo ${LINE} | awk {'print $6'}`
    GetServiceName ${PORT}
    RECORD[$INDEX]=`echo "$PORT ${SERVICE[0]} $CLIENT_ADDRESS $STATE"|\
        awk {'printf("%-12s%-20s%-25s%-20s",$1,$2,$3,$4)'}`
    INDEX=`expr $INDEX + 1`
done
INDEX=0
while [ "${RECORD[$INDEX]}" != "" ]
do
    echo "${RECORD[$INDEX]}" >> ${REPORT_FILE}
    INDEX=`expr $INDEX + 1`
done
NO_CONNECTIONS=`cat -n ${TEMP_FILE} | tail -1 | awk {'print $1'}`
echo "\n\nTotal number of connections is ${NO_CONNECTIONS}" \
    >> ${REPORT_FILE}
view ${REPORT_FILE}
PrintFile ${REPORT_FILE}
}
################################################################################
# Name     : DisplayRoutingTableInformation
#
```

```
# Overview : The function displays routing table information.
################################################################################
DisplayRoutingTableInformation ()
{
DisplayMessage I "${WORKING}" "N"
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Routing Table Information  on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "              $HEADER"     > ${REPORT_FILE}
echo "              $UNDERSCORE" >> ${REPORT_FILE}
echo "
Flags
=====
U Up
H The route is to a host rather than a network.
G The route is to a gateway.
D The route was created dynamically by a redirect.
M The route has been modified by a redirect.
L The link-level address is present in the route entry.
" >> ${REPORT_FILE}
netstat -nr >> ${REPORT_FILE}
view ${REPORT_FILE}
}
################################################################################
# Name     : ShowRouteForRemoteHost
#
# Overview : The function displays the route for a remote host.
################################################################################
ShowRouteForRemoteHost ()
{
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
# get host name
while true
do
    clear
    echo "Enter hostname (logical or IP address, l=list of values ):\c"
    read HOST
    case $HOST in
          "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
         l|L ) DisplayListOfValues "HOST" ;
               if [ "${SELECTED_VALUE}" = "" ]
               then
                   :  ;
               else
                   HOST="${SELECTED_VALUE}" ;
                   break ;
                fi ;;
           * ) break ;;
    esac
done
```

```
DisplayMessage I "${WORKING}" "N"
# ping the host
> ${TEMP_FILE}
# ping -R -c4 ${HOST} > ${TEMP_FILE} 2>&1
clear
traceroute ${HOST} | tee -a ${TEMP_FILE}
if [ $? -ne 0 ]
then
    vi  $TEMP_FILE
    DisplayMessage E "${NO_ROUTE}"
    OS_ERM=`grep "NOT FOUND" ${TEMP_FILE}`
    DisplayMessage E "${OS_ERROR}"
    return ${FALSE}
fi
# check the output file for loss of packets
# if ! grep "0% packet loss" ${TEMP_FILE} > /dev/null 2>&1
# then
#     DisplayMessage E "${PACKET_LOSS}"
#     return ${FALSE}
# fi
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Route Details for $HOST on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "             $HEADER"     > ${REPORT_FILE}
echo "             $UNDERSCORE" >> ${REPORT_FILE}
# CURDIR=`pwd`
# cd /tmp
# csplit -f ${TEMP_FILE_1} ${TEMP_FILE} /RR:/  /bytes/
# sed s/RR://  ${TEMP_FILE_1}01 1>> ${REPORT_FILE} 2> /dev/null
# cat ${TEMP_FILE_1}01 >> ${REPORT_FILE}
cat ${TEMP_FILE} >> ${REPORT_FILE}
view ${REPORT_FILE}
# cd $CURDIR
}
################################################################################
# Name     : DisplayNotesOnIPAddresses
#
# Overview : Displays detail notes on IP address.
################################################################################
DisplayNotesOnIPAddresses ()
{
HEADER="Analysis of IP Address"
FormatUnderscores "${HEADER}"
echo "          $HEADER"     > ${REPORT_FILE}
echo "          $UNDERSCORE" >> ${REPORT_FILE}
cat <<! >> ${REPORT_FILE}

The IP addressing scheme
------------------------
An Internet Protocol address is a unique 32-bit number assigned to a
```

host and used for all communication with that host.

The IP address hierarchy

Each 32-bit IP address has two parts: a prefix and a suffix. The prefix identifies the physical network to which the computer is attached and the suffix identifies the individual computer on that network.

Classes of IP address

There are three principal classes of address, and each class has a different size prefix and suffix. The first four bits of an address determine the class to which it belongs and specify how the remainder of the address is divided into prefix and suffix.

Division of the IP address space

Address class	Bits in prefix	Maximum number of networks	Bits in suffix	Maximum number of hosts per network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

Computing the class of an address

First four bits of address	Table index (in decimal)	Address class
0000	0	A
0001	1	A
0010	2	A
0011	3	A
0100	4	A
0101	5	A
0110	6	A
0111	7	A
1000	8	B
1001	9	B
1010	10	B
1011	11	B
1100	12	C
1101	13	C
1110	14	D
1111	15	D

Classes and dotted decimal notation

```
-----------------------------------

Range of decimal values found in the first octet of each address class:

Class    Range of values
  A        0 through 127
  B      128 through 191
  C      192 through 223
  D      224 through 239
  E      240 through 255

!
view ${REPORT_FILE}
PrintFile "${REPORT_FILE}"
}
################################################################################
# Name     : ListPortsInUse
#
# Overview : Displays all ports in use by TCP/IP.
################################################################################
ListPortsInUse ()
{
DisplayMessage I "${WORKING}" "N"
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Report on ports in use on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "   $HEADER" > ${REPORT_FILE}
echo "   $UNDERSCORE" >> ${REPORT_FILE}
netstat -n | grep "tcp" > ${TEMP_FILE}
sed s/*.*/UNKNOWN/ ${TEMP_FILE} > ${TEMP_FILE_1}
> ${TEMP_FILE}
cat ${TEMP_FILE_1} | while read LINE
do
    PORT=`echo ${LINE} | awk {'print $4'} | cut -d'.' -f5`
    echo $PORT >> ${TEMP_FILE}
done
cat ${TEMP_FILE} | sort -n -u > ${TEMP_FILE_1}
cat ${TEMP_FILE_1} >> ${REPORT_FILE}
view ${REPORT_FILE}
}
```

This article concludes in next month's issue of *AIX Update* with the remainder of the code for **mnd.sh**.

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

HTML documentation for SSA disks

This article describes a set of shell scripts that generate HTML documentation for SSA disks. The task of identifying disks in SSA cabinets and generating documentation for them is essential for the orderly maintenance of AIX systems. For example, when disk drives fail and are replaced, volume groups may need reorganizing and extra disks may need to be added to cabinets. In such instances, we have to update our documentation manually. An automated way to maintain this documentation is required, and the scripts in this article are designed not only to carry out this task but also to afford us a certain amount of flexibility in doing so. This acknowledges the fact that disk layouts change on a regular basis. The other main objective in writing these scripts is to allow a regular, scheduled, or manual run to take place to ensure that the documentation is up-to-date.

This article does not explain SSA cabling or design or discuss any new additions to the SSA world. It restricts itself to the task of producing HTML documentation for an actual set-up on an RS/6000 system. For further explanation of SSA disks, please refer to the IBM documentation. All the scripts take into account the fact that disks may be cabled to multiple systems using normal SSA cabling techniques. The scripts also produce a file in CSV format with information on just the disks that are used by the system running the script. This file can be loaded into spreadsheets for further analysis, which can be useful when there is a need to work on just one system.

MANUAL DETECTION OF SSA DISKS

Before the documentation can be automated, it is necessary to map *pdisk* numbers to physical locations in SSA cabinets manually. It is then necessary to update this information only when *pdisk* numbers change, which can be very rare if the procedure is followed to keep *pdisk* numbers when disks are changed (see the SSA documentation for details on how to do this). A key part of the identification process is the command **ssaidentify**, which can usually be found in */usr/sbin* after drivers for SSA disks have been installed (the file in */usr/sbin* is normally just a link to the actual command in */usr/ssa/disk/bin*; there

are numerous SSA disk utilities in this directory, and these are used extensively in the scripts).

For simplicity's sake, the documentation splits SSA cabinets into 'front' and 'back', with both containing the eight slots that are normally found in SSA cabinets. Once each disk is identified, the information is stored in a file, a copy of which is stored on all systems that contain the SSA disks. This file is used by the **buildtray** script, which we'll come to later. The format of the file and the naming convention used are essential to the successful running of the scripts.

NAMING CONVENTION FOR THE FILES

The naming convention adopted is shown in the example below.

```
pva.front_cabinet
```

The prefix ('pva' in this case) is used to control the order in which the list is produced; 'pva' is the front cabinet and 'pvb' the rear cabinet.

'front_cabinet' is the title of the document. All underscores ('_') are replaced with spaces in the title, so the description can be as precise as needed.

THE FILE FORMAT

The example below shows the content of *pva.front_cabinet*.

```
13 14 15 16 17 21 22 10
30 31 32 40 -- 34 41 1
```

This file contains *pdisk* numbers that are identified manually using the command **ssaidentify**. Each drawer in a cabinet appears in the file in order from the top down. So, in our example, tray two is on line 1 and the bottom tray one is on line 2. A double '-' indicates an empty slot.

LOCATION OF THE FILES

The variable *DISK_DIR* in the **buildtray** script points to the directory where the files that are created are stored – you should set this to the directory you want to use. The variable *DATA* in the **buildtray** script should point to your data directory.

The script below can help you to coordinate the process of identifying your SSA disks.

SCRIPT TO IDENTIFY SSA DISKS

```
#!/bin/ksh
# Script to identify SSA disks.
# Written by Robert Russell.
# The script detects SSA disks in hdisk order; press the return key
# to move to the next disk once the pdisk number as been noted
for i in `lscfg -l hdisk* |grep SSA |sort -tk +1 -n|awk '{print $1}'`
do
PD=`ssaxlate -l $i`
echo $PD
ssaidentify -l $PD -y
read
ssaidentify -l $PD -n
done
```

If you run this script as *root*, the script prints out the *pdisk* number and starts the identification process. To identify a disk, check which SSA disk has its identification light flashing. It's worth creating a table with eight columns (one for each drawer) in preparation for recording the number of the *pdisk* in a specific drawer on either the front or back of the SSA cabinet. When the disk is identified, write down the *pdisk* number in the corresponding cell on the table – this will make it easier to create the disk files. The identification process progresses from left to right as you look at the cabinet from either the front or back. (We chose this format so we could print off the latest copy of the file and easily match its layout to that of the SSA cabinet.)

The latest SSA disks and drivers have made the process of identifying SSA disks much easier. However, our own disks are mostly older ones – and are likely to be so for the foreseeable future – so there is no way around this manual stage for us.

DESCRIPTION OF THE SCRIPTS

There are four shell scripts that produce the documentation; they are as follows:

1 **buildtray** produces an HTML document with details of the SSA disk cabinet.

2 **creator** produces an HTML document with details of physical disks.

3 **creatorlv** produces an HTML document with details of logical volumes.

4 **creatorvg** produces an HTML document with details of volume groups.

All HTML files have links to other files where you can trace volume groups, logical volumes, and filesystems to actual physical disks in the SSA cabinet.

All scripts should be in the same directory and the HTML output files should also be in one directory. The *DATA* and *DISK_DIR* variables should be set to reflect your chosen directories, as discussed earlier.

THE BUILDTRAY SCRIPT

To run this script successfully, it is essential to identify your SSA disks in your cabinets correctly. If there is any doubt, check the documentation thoroughly.

Note the use of the continuation character, '➤', in the code below to indicate a formatting line break that's not present in either the original source code or the code that can be downloaded from Xephon's Web site (*http://www.xephon.com/aixupdate.html*).

BUILDTRAY

```
#!/bin/ksh
#########################################################
#            Written By Robert Russell                  #
#                                                       #
#               Document SSA DISKS                      #
#                                                       #
#           HTML and CSV files produced                 #
#########################################################
#set -x
integer COUNT
integer C
integer SSANUM
integer NUM
DATA=/home/it032x/html
```



```
DISK_DIR=$DATA/disk
OUTFILE=$DATA/`hostname`SSA.html
WORKFILE=$DATA/work.tmp
COLFILE=$DATA/colour
CSV=$DATA/disks.csv
HOST=$DATA/otherhost
ERR=$DATA/error
TRAY=""
TITLE=""
i=" "
LINK=""
LINKREF=""
LINKEXT=""
>$CSV
print "HOST,CABINET,TRAY,PDISK,HDISK,SIZE,VG/HOST,S/N,MODEL,ROS,
➤  PP SIZE,FREE PP,TOTAL PP,USED PP,UNIT">$CSV
DEFCOL="white"
COLCODE=`grep $DEFCOL $COLFILE |awk '{print $2}'`
COLPASS=$COLCODE
SSANUM=`lsdev -Cc adapter |grep ssa |wc -l `
C=1
while [ C -le SSANUM ]
do
    LOOP[$C]=""
    LOOPLINE[$C]=""

C=$C+1
done
>$OUTFILE
colgrab()
{
COLPASS=`grep $1 $COLFILE |awk '{print $2}'`
if [[ "$?" != 0 ]]
then
    COLPASS=$DEFCOL
fi
}
tray_build()
{
print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=3></FONT>
➤  <FONT FACE=\"Arial\"></P><A NAME=\"$i$TRAY\"></A>
➤  $TITLE $TRAY</A>">>$OUTFILE
print "<TABLE BORDER CELLSPACING=1 CELLPADDING=1 WIDTH=700 >">>$OUTFILE
}
row_build()
{
echo ""
W="10%"
COLCODE=$COLPASS
print "<TD BGCOLOR=\"#$COLCODE\" WIDTH=\"$W\" ALIGN=\"CENTER\">
```

```
➤  <B>$TNAME</B></TD>">>$OUTFILE
for t in `echo $PASS`
do
    if [[ "$LINK" = "N" ]]
    then
        print "<TD BGCOLOR=\"#$COLCODE\" WIDTH=\"$W\"
        ➤  ALIGN=\"CENTER\">$t</TD>">>$OUTFILE
    else
        if [[ "$t" != "FREE" ]]
        then
            if [[ "$PASS" != "$LINEVG" ]]
            then
                print "<TD BGCOLOR=\"#$COLCODE\" WIDTH=\"$W\"
                ➤  ALIGN=\"CENTER\"><A NAME=\"$LINKEXT$t\"></A>
                ➤  <A HREF=\"$LINKREF.html#$LINKEXT$t\">$t</A>
                ➤  </TD>">>$OUTFILE
            else
                grep $t $HOST >/dev/null 2>&1
                if [[ "$?" = "0" ]]
                then
                    print "<TD BGCOLOR=\"#$COLCODE\" WIDTH=\"$W\"
                    ➤  ALIGN=\"CENTER\"><A NAME=\"$LINKEXT$t\"></A>
                    ➤  <A HREF=\"${t}SSA.html#$i$TRAY\">$t</A>
                    ➤  </TD>">>$OUTFILE
                else
                    print "<TD WBGCOLOR=\"#$COLCODE\" IDTH=\"$W\"
                    ➤  ALIGN=\"CENTER\"><A NAME=\"$LINKEXT$t\"></A>
                    ➤  <A HREF=\"$LINKREF.html#$LINKEXT$t\">$t</A>
                    ➤  </TD>">>$OUTFILE
                fi
            fi
        else
            print "<TD BGCOLOR=\"#$COLCODE\" WIDTH=\"$W\"
            ➤  ALIGN=\"CENTER\">$t</TD>">>$OUTFILE
        fi
    fi
done
print "</TR>">>$OUTFILE
}
font()
{
echo ""
}
page_title()
{
print "<B><P ALIGN=\"CENTER\"></FONT><FONT FACE=\"Symbol\"
➤  SIZE=4></FONT><FONT FACE=\"Arial\"></P><A NAME=\"$TITLE\"></A>
➤  <P ALIGN=\"CENTER\">$TITLE</P></A>">>$OUTFILE
print "<P></P>">>$OUTFILE
}
```

```
host_check()
{
D="N"
for h in `cat $HOST`
do
        TEMP=`rsh $h cat $DISK_DIR/$i |tail -$COUNT |head -1
        ➤  |cut -f$NUM -d" " `
        TDISK=`rsh $h ssaxlate -l pdisk$TEMP`
        rsh $h lspv $TDISK >/dev/null 2>&1
        if [[ "$?" != "0" ]]
        then
            if [[ "$D" = "N" ]]
            then
                VG="FREE"
            fi
        else
            VG="$h"
            D="Y"
        fi
done
}
#MAIN SCRIPT START
for i in `ls $DISK_DIR`
do
COUNT=`cat $DISK_DIR/$i|wc -l`
TITLE=`hostname`" "`echo $i|cut -f2 -d\.|sed "s/_/ /g"`
page_title
echo $TITLE
echo $TRAY
echo $DISK_DIR/$i
echo $COUNT
while [ $COUNT -gt 0 ]
do
cat $DISK_DIR/$i| tail -$COUNT |head -1 | read DISK
    TRAY="TRAY$COUNT"
    NUM=0
    for pd in `echo $DISK`
    do
        NUM=$NUM+1
        if [[ "$pd" != "--" ]]
        then
        d=`ssaxlate -l pdisk$pd|sed "s/hdisk//g"`
        PD="pdisk$pd"
        MODEL=`lscfg -vl $PD|grep Model|cut -f7 -d.'`
        case "$MODEL" in
            DFHCC4B1)
                SIZE="4.5GB"
            ;;
            DCHC09B1)
                SIZE="9.1GB"
```

```
    ;;
    DGHC09B)
        SIZE="9.1GB"
    ;;
    DGHC18B)
        SIZE="18.2GB"
    ;;
    *)
        SIZE="UNKN"
    ;;
esac
SER=`lsattr -El $PD -aconnwhere_shad|awk '{print $2}'
➤  |cut -c5-12`
ROS=`lscfg -vl $PD|grep ROS|cut -f13 -d.'`
lspv hdisk$d >/dev/null 2>&1
if [[ "$?" != "0" ]]
then
    if [[ -f $HOST ]]
    then
        host_check
    else
        VG="FREE"
    fi
else
    lspv hdisk$d>$WORKFILE
    VG=`cat $WORKFILE |grep "VOLUME GROUP" |awk '{print $6}'`
    cat $WORKFILE |grep "PP SIZE"|awk '{print $3,$4}'
    ➤  |read PP_SIZE UNIT
    FREE_PP=`cat $WORKFILE |grep "FREE PP"|awk '{print $3}'`
    TOTAL_PP=`cat $WORKFILE |grep "TOTAL PP" |awk '{print $3}'`
    USED_PP=`cat $WORKFILE |grep "USED PP" |awk '{print $3}'`
    HOST_CAB=`echo $TITLE|sed "s/ /,/"`
    echo "$HOST_CAB,$TRAY,$PD,hdisk$d,$SIZE,$VG,$SER,$MODEL,
    ➤  $ROS,$PP_SIZE,$FREE_PP,$TOTAL_PP,$USED_PP,$UNIT">>$CSV
fi
C=1
for ssa in `lsdev -Cc adapter |grep ssa|awk '{print $1}' `
do
    >$ERR
    LOOP[$C]=${LOOP[$C]}" `ssaconn -l $PD -a $ssa 2>$ERR|awk
    ➤  '{print "\("$3"-"$4"\)\<\/br\>\("$5"-"$6"\)" }'`"
    if [[ -s $ERR ]]
    then
        LOOP[$C]=${LOOP[$C]}" n/a"
    fi
    C=$C+1
done
LINESIZE=$LINESIZE" "$SIZE
LINEDISK=$LINEDISK" "$d
LINEVG=$LINEVG" "$VG
```

```
        LINEPD=$LINEPD" "$PD
        LINESER=$LINESER" "$SER
        LINEMODEL=$LINEMODEL" "$MODEL
        LINEUSED=$LINEUSED" "$USED_PP
        LINEROS=$LINEROS" "$ROS
        else
            LINESIZE=$LINESIZE" FREE"
            LINEDISK=$LINEDISK" FREE"
            LINEVG=$LINEVG" FREE"
            LINEPD=$LINEPD" FREE"
            LINESER=$LINESER" FREE"
            LINEMODEL=$LINEMODEL" FREE"
            LINEROS=$LINEROS" FREE"
            LINEUSED=$LINEUSED" FREE"
            C=1
            for ssa in `lsdev -Cc adapter |grep ssa|awk '{print $1}' `
            do
                LOOP[$C]=${LOOP[$C]}" FREE"
                C=$C+1
            done
        fi
    done
        tray_build
        LINK="N"
        echo $LINEDISK
        TNAME="HDISK"
        LINK="Y"
        LINKREF="`hostname`lspvl"
        LINKEXT="hdisk"
        PASS=$LINEDISK
        colgrab white
        row_build
        LINK="N"
        echo $LINESIZE
        TNAME="SIZE"
        PASS=$LINESIZE
        row_build
        echo $LINEVG
        LINK="Y"
        TNAME="VG/host"
        LINKREF="`hostname`lsfs"
        LINKEXT=""
        PASS=$LINEVG
        row_build
        LINK="N"
        echo $LINEPD
        TNAME="PDISK"
        PASS=$LINEPD
        row_build
        echo $LINESER
```

```
        TNAME="S/N"
        PASS=$LINESER
        colgrab grey
        row_build
        echo $LINEMODEL
        TNAME="MODEL"
        PASS=$LINEMODEL
        row_build
        echo $LINEROS
        TNAME="ROS"
        PASS=$LINEROS
        row_build
        C=1
        for ssa in `lsdev -Cc adapter |grep ssa|awk '{print $1}' `
        do
            echo ${LOOP[$C]}
            TNAME="$ssa (a)</br>(b)"
            PASS=${LOOP[$C]}
            colgrab cyan
            row_build
            C=$C+1
        done
        print "</TABLE>">>$OUTFILE
        print "</br>">>$OUTFILE
COUNT=$COUNT-1
echo $COUNT
        LINESIZE=""
        LINEDISK=""
        LINEVG=""
        LINEPD=""
        LINESER=""
        LINEMODEL=""
        LINEROS=""
        C=1
        for ssa in `lsdev -Cc adapter |grep ssa|awk '{print $1}' `
        do
            LOOP[$C]=""
            C=$C+1
        done
done
print "</br></br>">>$OUTFILE
#print "<P> </P>">>$OUTFILE
#print "<P> </P>">>$OUTFILE
done
```

SSA DISK MODELS

The following code can be used to establish the size and model of SSA disks. The size of the disk is based on the model, and you can check

your disk's model using the command:

```
lscfg -vl pdisk
```

where *pdisk* is the disk number. You can add disk models manually to the code by following the format of *NEW MODEL* in the example below, replacing *NEW MODEL* with the actual model number and *NEWGB* with the size of the disk.

```
MODEL=`lscfg -vl $PD|grep Model|cut -f7 -d.'`
case "$MODEL" in
    DFHCC4B1)
    SIZE="4.5GB"
    ;;
    DCHC09B1)
    SIZE="9.1GB"
    ;;
    DGHC09B)
    SIZE="9.1GB"
    ;;
    DGHC18B)
    SIZE="18.2GB"
    ;;
    NEW MODEL)
    SIZE="NEWGB"
    ;;
    *)
    SIZE="UNKN"
    ;;
```

COMPANION FILES

Colour file

The script uses a file to determine the background colour of cells in the HTML table. It is referenced as follows:

```
COLFILE=$DATA/colour
```

This file, whose default name is *colour*, needs to be created before the **buildtray** script is first run. Its contents are as follows:

```
red FF0000
green 00FF00
blue 0000FF
black 000000
white FFFFFF
grey 888888
```

```
yellow FFFF00
cyan 00FFFF
```

These are just standard HTML colour references.

Other host file

The script uses a file to determine which other hosts may have access to the same SSA disks. It is referenced as follows:

```
HOST=$DATA/otherhost
```

If there are multiple hosts in the SSA cable loops, then this file, whose default name is *otherhost*, needs to be in place before the **buildtray** script is first run. Its contents must conform with the following format:

```
hostname2
hostname3
```

For the script to work, each host listed should allow remote shell sessions to run. If security forbids this, then another method needs to be implemented to support this functionality in the scripts.

CSV file

The **buildtray** script builds a comma-separated file that can be imported by spreadsheet programs for analysis. This file is created automatically when the script is run. Its default name is *disks.csv* and it's referenced as follows.

```
CSV=$DATA/disks.csv
```

Two other files are created when the script is run: a working file and an error file that can be used for troubleshooting.

buildtray HTML File

The main output of the **buildtray** script is an HTML file that's referenced as follows:

```
OUTFILE=$DATA/`hostname`SSA.html
```

As you can see, the hostname is part of the file name and 'SSA.html' is appended to the end. It's important that this file name isn't changed, otherwise the links won't work.

CREATOR SCRIPT

```
#!/bin/ksh
########################################
#      Written By Robert Russell       #
#   Created for HTML documentation     #
########################################
#integer COUNTinteger TCOUNT=0OUTFILE=/home/it032x/html/
➤  `hostname`lspvl.html SSAF="`hostname`SSA"LVF="
➤  `hostname`lslvl"HN=`hostname`
>$OUTFILE
print "<P></P>">>$OUTFILE
print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=4></FONT>
➤  <FONT FACE=\"Arial\"></P><A NAME=\"Disk Information\"></A>
➤  $HN DISK INFORMATION</A>">>$OUTFILE
print "<P></P>">>$OUTFILE
for i in `lscfg -l hdisk* |grep SSA |sort -tk +1 -n|awk '{print $1}'`
do
    T=`echo $i |sed "s/hdisk//g"`
    print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=3></FONT>
    ➤  <FONT FACE=\"Arial\"></P><A NAME=\"$i\"></A><A HREF=
    ➤  \"$SSAF.html#$i\">$HD hdisk$T</A>">>$OUTFILE
    print "</B></B></FONT><FONT FACE=\"Courier New\" SIZE=2>">>$OUTFILE
    print "<TABLE BORDER CELLSPACING=1 CELLPADDING=1
    ➤  WIDTH=685>">>$OUTFILE
    lspv -l $i |grep -v ":" |sed "s/\([0-9]\)\//\1<\/TD><TD>\//
s/   */<\/TD><TD>/g
2;\$s/\(^[A-Za-z0-9][A-Za-z0-9]*\)/<A HREF=\"$LVF.html#\1\">\1<\/A>/
s/^/<TD>/
s/$/<\/TD><\/TR>/">>$OUTFILE
    print "</TABLE>">>$OUTFILE
done
```

creator HTML File

The main output of the **creator** script is an HTML file that is referenced as follows:

```
OUTFILE=/home/it032x/html/`hostname`lspvl.html
```

Again, the hostname is an integral part of the file name, with ‘lspvl.html’ appended to it. As before, it’s important that the file name isn’t changed. The directory can be changed, and must be set to the same one in which all the other HTML files are stored.

The code below is important and care should be taken to ensure it’s input correctly. It uses the **sed** command and each line must be separated from the succeeding one by a carriage return only, otherwise

syntax errors occur. All scripts that use **sed** in this way can also be implemented using **sed -f** and creating separate files. I chose this method as it keeps all the information in one place.

```
lspv -l $i |grep -v ":" |sed "s/\([0-9]\)\//\1<\/TD><TD>\//
s/   */<\/TD><TD>/g
2;\$s/\(^[A-Za-z0-9][A-Za-z0-9]*\)/<A HREF=\"$LVF.html#\1\">\1<\/A>/
s/^/<TD>/
s/$/<\/TD><\/TR>/">>$OUTFILE
```

CREATORLV SCRIPT

```
#!/bin/ksh
#########################################
#      Written By Robert Russell     #
#   Created for HTML documentation   #
#########################################
#integer COUNT
integer T
COUNT=0
OUTFILE=/home/it032x/html/`hostname`lslvl.html
SSAF="`hostname`SSA"
LVF="`hostname`lslvl"
PVF="`hostname`lspvl"
HN=`hostname`
>$OUTFILE
print "<P></P>">>$OUTFILE
print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=4></FONT>
➤  <FONT FACE=\"Arial\"></P><A NAME=\"LV Information\"></A>
➤  $HN LV INFORMATION</A>">>$OUTFILE
print "<P></P>">>$OUTFILE
for v in `lsvg|grep -v rootvg`
do
    for i in `lsvg -l $v|awk '{print $1}'|egrep -v ":|LV"`
    do
    #print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=3></FONT>
    ➤  <FONT FACE=\"Arial\"></P><A NAME=\"$i\"></A><A HREF=
    ➤  \"$SSAF.html#$i\">$i</A>">>$OUTFILE
    print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=3></FONT>
    ➤  <FONT FACE=\"Arial\"></P><A NAME=\"$i\"></A>
    ➤  $HN $i</A>">>$OUTFILE
    print "</B></B></FONT><FONT FACE=\"Courier New\" SIZE=2>">>$OUTFILE
    print "<TABLE BORDER CELLSPACING=1 CELLPADDING=1 WIDTH=685>"
    ➤  >>$OUTFILE
    lslv -l $i |grep -v "$i" |sed "s/   */<\/TD><TD>/g
2;\$s/\(^[A-Za-z0-9][A-Za-z0-9]*\)/<A HREF=\"$PVF.html#\1\">\1<\/A>/
s/^/<TD>/
s/$/<\/TD><\/TR>/">>$OUTFILE
    print "</TABLE>">>$OUTFILE
```

```
done
done
```

creatorlv HTML file

The **creatorlv** script, like **creator**, outputs an HTML file that is referenced as follows:

```
OUTFILE=/home/it032x/html/`hostname`lslvl.html
```

As with the other scripts, the hostname is an integral part of the file name, which also has an ‘lslvl.html’ appended to it. Again, the file name must not be changed for links to work. The directory can be changed and must be the same one that’s used to store all the other HTML files.

CREATORVG SCRIPT

```
#!/bin/ksh
#########################################
#      Written By Robert Russell      #
#   Created for HTML documentation    #
#########################################
#integer COUNT
integer T
COUNT=0
OUTFILE=/home/it032x/html/`hostname`lsfs.html
SSAF="`hostname`SSA"
LVF="`hostname`lslvl"
HN=`hostname`
>$OUTFILE
print "<P></P>">>$OUTFILE
print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=4></FONT>
➤  <FONT FACE=\"Arial\"></P><A NAME=\"Volume Group Filesystems\"></A>
➤  $HN Volume Group Filesystems</A>">>$OUTFILE
print "<P></P>">>$OUTFILE
for i in `lsvg |grep -v rootvg`
do
    print "<B><P></FONT><FONT FACE=\"Symbol\" SIZE=3></FONT>
    ➤  <FONT FACE=\"Arial\"></P><A NAME=\"$i\">$HN Filesystems in
    ➤  Volume Group $i</A>">>$OUTFILE
    print "</B></B></FONT><FONT FACE=\"Courier New\" SIZE=2>">>$OUTFILE
    print "<TABLE BORDER CELLSPACING=1 CELLPADDING=1 WIDTH=685>"
    ➤  >>$OUTFILE
    print "<TD>Logical Volume</TD><TD>Filesystem Mount Point</TD><TD>
    ➤  Auto Mount</TD></TR>">>$OUTFILE
    for v in `lsvgfs $i`
    do
```

```
    lsfs -l $v|grep -v "Name" |  read LV S1 FS TYPE SIZE OPTIONS
    ➤  AUTO ACC
    REF=`basename $LV`
    print "<TD><A HREF=$LVF.html#$REF>$LV</A></TD><TD>$FS</TD>
    ➤  <TD>$AUTO</TD></TR>">>$OUTFILE
done
    print "</TABLE>">>$OUTFILE
done
```

creatorvg HTML File

The **creatorvg** script outputs an HTML file that's referenced as follows:

```
OUTFILE=/home/it032x/html/`hostname`lsfs.html
```

The hostname comprises the first part of the file name, which has an 'lsfs.html' appended to it. The file name must not be changed, though the directory can be changed and must be the same one in which other HTML files are stored.

AUTOMATION AND SCHEDULING THE HTML DOCUMENTATION

The following script can be used to tie the scripts together, enabling you to issue just one command to create all the documentation.

AUTOMATION SCRIPT

```
#!/bin/ksh
#written by Robert Russell to create HTML documentation files
DATA=/home/it032x/html
$DATA/buildtray
$DATA/creator 2>/dev/null
$DATA/creatorlv 2>/dev/null
$DATA/creatorvg
```

Once created, this can be scheduled using **cron** to update your documentation regularly. Alternatively, use it after managing SSA disks, volume groups, or filesystems.

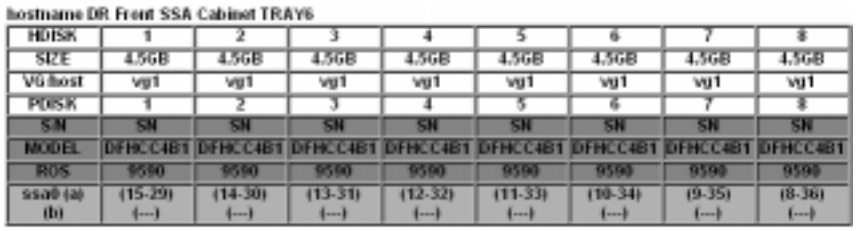
HTML DOCUMENTATION FILES

Each time the above script is run, four HTML files that contain SSA documentation are created.

hostnameSSA.html

This file contains front and back views of the SSA cabinet (Figure 1). The interpretation of the rows is as follows:

HDISK	The number of the disk, which is used when allocating disks to volume groups.
SIZE	The capacity of the disk in GB, which is determined from the model number.
VG/host	The name of the volume group or host to which the disk is assigned.
S/N	The disk's serial number.
MODEL	The disk's model number.
ROS	The disk's microcode level.
ssax (a)(b)	Cable information, indicating the 'distance' between the controller and the disk. (Refer to the SSA manuals for more information on cabling.)



hostname DR Front SSA Cabinet TRAY6

HDISK	1	2	3	4	5	6	7	8
SIZE	4.5GB	4.5GB	4.5GB	4.5GB	4.5GB	4.5GB	4.5GB	4.5GB
VG/host	vg1	vg1	vg1	vg1	vg1	vg1	vg1	vg1
PDISK	1	2	3	4	5	6	7	8
S/N	SN	SN	SN	SN	SN	SN	SN	SN
MODEL	DFHCC4B1	DFHCC4B1	DFHCC4B1	DFHCC4B1	DFHCC4B1	DFHCC4B1	DFHCC4B1	DFHCC4B1
ROS	9590	9590	9590	9590	9590	9590	9590	9590
ssa0 (a) (b)	(15-29) (---)	(14-30) (---)	(13-31) (---)	(12-32) (---)	(11-33) (---)	(10-34) (---)	(9-35) (---)	(8-36) (---)

Figure 1: hostnameSSA.html

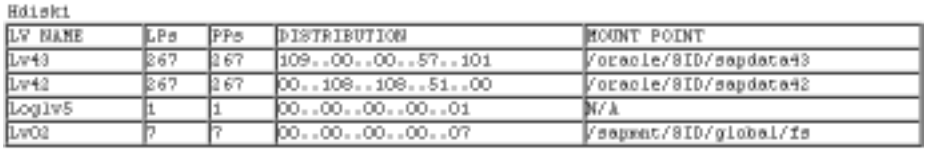
hostnamelspvl.html

This presents the following information about each *hdisk* attached to the system (see Figure 2).

LV name:	Logical volume name.
LPs	Logical partitions assigned to the logical volume.
PPs	Physical partitions assigned to the logical volume.

Distribution Distribution of partitions on the disk.

Mount point Mount point on the system.



Hdisk1

LV NAME	LPs	PPs	DISTRIBUTION	MOUNT POINT
Lv43	267	267	109..00..00..57..101	/oracle/8ID/sapdata43
Lv42	267	267	00..108..108..51..00	/oracle/8ID/sapdata42
Loglv5	1	1	00..00..00..00..01	N/A
Lv02	7	7	00..00..00..00..07	/sapmnt/8ID/global/fs

Figure 2: hostnamelspvl.html

hostnamelslvl.html

This presents the following information for each logical volume on a system (see Figure 3).

PV The hdisk that hosts the logical volume.

Copies The number of partitions on the disk.

In band This provides an indication of how well partitions are distributed, given the requirements.

Distribution The distribution of the partitions on the disk.

PV	COPIES	IN BAND	DISTRIBUTION
Hdisk9	268:000:000	40%	000:105:105:052:000

Figure 3: hostnamelslvl.html

hostnamelsfs.html

This presents the following information for each volume group on a system (see Figure 4).

Logical volume The logical volume name.

Filesystem mount point Filesystem allocated to the logical volume.

Auto mount Determines whether the filesystem is remounted when the system is restarted.

hostname Filesystems in Volume Group vg3

Logical Volume	Filesystem Mount Point	Auto Mount
/dev/lvarch	/oracle/SID/saparch	yes

Figure 4: hostnamelsfs.html

HTML LINKS

All HTML files generated need to be in the same directory for the links to work – all links are relative and no full path names are specified, which means the links refer to files in the same directory. The links are as follows:

- *hostnameSSA.html*
  - *HDISK* links to *hostnamelspv.html*
  - *VG/host* links to either *hostnamelsfs.html* or *hostnameSSA.html*, depending on the system to which the SSA disk is allocated. (For the *host* link to work, the corresponding *hostnameSSA.html* file should be in the same directory.)
- *hostnamelsfs.html*
  - The logical volume links to *hostnamelslvl.html*.
- *hostnamelslvl.html*
  - *PV* links to *hostnamelspvl.html*.
- *hostnamelspvl.html*
  - The logical volume name links to *hostnamelslvl.html*.
  - The *hdisk* links to *hostnameSSA.html*.

CSV DOCUMENTATION FILE

The *disks.csv* file is created by default whenever the **buildtray** script is run. The file contains the following information on disks allocated to the system:

Host	Hostname
Cabinet	SSA cabinet (front or back)
Tray	Drawer/tray
Pdisk	*Pdisk* number
Hdisk	*Hdisk* number
Size	Size of disk
VG/host	Volume group and host
S/N	Serial number
Model	Disk model
ROS	Microcode level
PP size	Physical partition size
Free PP	Free physical partitions
Total PP	Total physical partitions
Used PP	Used physical partitions
Unit	Measure of physical partition size.

Robert Russell (UK) © Xephon 2000

Help with large printing systems

Printing is one of the most problematic areas for system administrators. On large systems that serve users across many sites, the number of network/remote printers that need to be managed can be considerable.

While AIX manages the queues for you via the **qdaemon**, on systems with a large number of printers, the **lpstat** command can take a long time to complete for general queries about the print subsystem. If you know the name of a printer that's playing up (via the help desk etc), then you can always query the printer directly using the command **lpstat –p<queue_name>**.

However, it's always better to be proactive and try to find problems as early as possible, which is the motivation behind the script in this article. The script simply examines the */usr/spool/lpd/qdir* directory and counts the number of jobs queued for each printer. The script works for AIX Version 4 and will, I believe, work for Version 3.

THE SCRIPT

```
#!/bin/ksh
clear
echo "\n                    ##################################
\n"

ls /usr/spool/lpd/qdir  > /tmp/$$.printlist
wc -l /tmp/$$.printlist | read a rubbish
echo "There are $a print files to process/being processed"
if [[ $a -gt 0 ]]
then
  cat /tmp/$$.printlist | awk -F"$" '{ print $1 }' | awk -F":" '{ print
$2 }' >
/tmp/$$.printers
  sort -u /tmp/$$.printers > /tmp/$$.printeruniq
  echo "Files are queued for printers : \c"
  for j in `cat /tmp/$$.printeruniq`
  do
    searchfor="$j\$"
    grep -c $searchfor /tmp/$$.printers | read z
    echo $j " ("$z")  \c"
  done
fi
rm /tmp/$$.printeruniq /tmp/$$.printlist /tmp/$$.printers 2> /dev/null

echo "\n\n                    ##################################
\n"
```

Phil Pollard
Unix System Programmer (UK)

AIX news

IBM has announced HAGEO 2.2 and GeoRM 2.2 (Geographic Remote Mirror for AIX), which provide real-time mirroring of data between systems connected by local or point-to-point networks, bringing disaster recovery to RS/6000 wide-area clusters.

HAGEO responds to site and communication failures and provides facilities for site takeover. Tools are available for synchronizing data after an outage, configuration management, capacity planning, performance monitoring, and problem determination.

A GeoRM server at a geographically remote location can mirror data in real-time, allowing users to back up and recover data as needed. HAGEO extends HACMP's loosely coupled cluster technology by providing more access to data and applications and eliminating the site as a single point of failure.

Out now, HAGEO 2.2.0 starts at US$10,000 for a D5, rising to US$37,500 for a P5, while the cost of GeoRM 2.2.0 is US$7,000 for a D5, rising to US$18,000 for a P5.

For further information, contact your local IBM representative.

* * *

Computer Associates has released eTrust Access Control 5.0 for Unix, an uprated version of the company's security product that now supports AIX 4.3.2. The software enables Unix users to protect corporate data and applications using security policies based on role or group membership.

The new version has better management and administration features, allowing central control of the creation, distribution, and management of access policies, and can also be part of a Unicenter environment.

Out now, prices weren't announced.

For further information contact:
Computer Associates International, 1 Computer Associates Plaza, Islandia, NY 11788, USA
Tel: +1 516 342 5224
Fax: +1 516 342 5734
Web: http://www.cai.com

Computer Associates, Ditton Park, Riding Court Road, Datchet, Slough, Berkshire SL3 9LL, UK
Tel: +44 1753 577733
Fax: +44 1753 825464

* * *

IBM has released Version 3.5 of its WebSphere Application Server Advanced Edition, which supports the Java 2 Software Development Kit V1.2.2 across on all its platforms, including AIX. Java 2 SDK promises increased performance and expanded capabilities for improved dynamic and static Web content. Usability enhancements include improved installation and administration procedures.

An Apache-based HTTP server is also included, and this has additional security and performance enhancements, including a fast Common Gateway Interface and LDAP Client 3.1 support. Out now, prices are available on request from IBM.