65

AIX

March 2001

In this issue

update

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

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* can be downloaded from our Web site at http://www.xephon.com/aixupdate.html; you will need the user-id shown on your address label.

Editors

Trevor Eddolls and Richard Watson

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.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*, or you can download a copy from www.xephon.com/contnote.html.

Printed in England.

The ODM and what's in it (for us)

The ODM (Object Data Manager) is a piece of software embedded in AIX that manages the storage, updating, and retrieval of data concerning the RS/6000 system hardware and software configuration. As described in *AIX Update* Issue 37, November 1998, there are three installed directories where the database is stored – */usr/lib/objrepos*, */etc/objrepos*, and */usr/share/lib/objrepos*. An environment variable *ODMDIR* is used to state which one of the directories is to be used at any one time. A series of commands and C language routines are provided to access/update the data in the ODM.

Note: the ODM editor (*odme)*, referred to in Issue 37, was discontinued in AIX Version 4.

THE ODM STRUCTURE

The three directories holding the ODM have distinct versions of data, the more relevant of which are mentioned below:

- */usr/lib/objrepos* – holds the 'predefined' hardware databases for the system hardware, for example the Pd* files. PdDv holds the Predefined Hardware Device values (tok, etc); PdAt holds the Predefined System Attribute values. Also, it holds the 'usr' part of the software installation.
- */etc/objrepos* – holds the 'customized' hardware databases for the system, eg the Cu* files. CuDv holds the definitions for the installed hardware (initially built at boot-time); CuAt holds the customized setting for the running system, such as the number of processes allowed per user. Also, it holds the 'root' part of the software installation.
- */usr/share/lib/objrepos* – holds the 'share' part of the software installation.

There are several files within each of the directories that hold the data. The name of the file is also the name of the 'class'. Hence to query the

lpp class data, which holds information regarding the *installp* activity and state of installed products, simply issue the command *odmget lpp*.

THE ODM COMMANDS

I will deal only with the supplied commands for manipulating the ODM. However, there are equivalently-named C language routines.

- *odmcreate* – creates class definitions for a new class. A new file is created in the relevant ODMDIR specified directory, with the same name as the class name. C language structures (.h) can also be produced.
- *odmshow* – displays the C language class definition for a given class.
- *odmadd* – stores a data instance of a class.
- *odmget* – displays the contents of a class.
- *odmchange* – alters the contents of data within a class.
- *odmdelete* – removes an instance from within a class.
- *odmdrop* – removes the class and all its data.

Most, if not all, AIX commands that access data within the ODM use the C language equivalents of the above commands. However, it is easy to see the similarities between some AIX commands and the above ODM commands. For example, the values output from the commands *odmget -q "name=sys0" CuAt* and *lsattr -El sys0* are not too dissimilar.

SO, WHAT IS IN IT FOR US?

Theory over, on with the practice. The IBM manuals go on about Cinderella and her friends. This is all very well, but as system administrators/programmers, it is of limited use. Apart for rewriting some of the AIX commands as an exercise, there seems little reason for users to use the ODM. However, back in the 'good' old days of AIX 3.2, there was occasion to fiddle with the data as more often than not

installp would give up on a PTF and some 'rectifying' would be required. With AIX 4, the fiddling has become obsolete.

The data stored in the ODM should be 'smallish', ie not vast quantities, and should not change often. The MOTD is a reasonable example, but the format is not ideal and we have a file for that already. However, some general data can be stored in the ODM as described in the following example.

The example below shows how to create, store, update, and delete a class called *sysdata*, which will hold the following information:

- Hostname of the machine.
- The name of the application running on the machine.
- The physical location of the machine.
- The name and contact number of the system administrator for that machine.

First we need to decide where we are to store the data. Because all the machine-specific data is stored in */etc/objrepos*, we'll store it there. Hence:

```
export ODMDIR=/etc/objrepos
```

Next we need to create a file to hold our class definition. This file can be called anything, but should have the **.cre** extension. Our file is called **odm_def.cre**:

```
class  sysdata {
       char  Hostname[30];
       char  Application[40];
       char  Location[20];
       char  Contact[30];
};
```

To add the definition to the ODM, issue the command:

```
odmcreate odm_def
```

This will create the *sysdata* file in the */etc/objrepos* directory and create *.c* and *.h* files in the current directory.

Having set up the class definition we can now add the data for this particular machine. Again using any filename (eg **odm_data**), you create a class instance:

```
sysdata:
      Hostname = "aix.xephon.com"
      Application = "DNS and NIS server"
      Location = "In machine-room 1"
      Contact = "Anyone you can find x666"
```

Now to add this data into the ODM simply issue the command:

```
odmadd odm_data
```

Simple really, without having to trouble Cinderella. Extra things you may wish to do are update the data. This is easily done:

```
odmet sysdata > file_name
vi file_name
odmchange -o sysdata < file_name
```

And to remove the data entirely:

```
odmdrop sysdata
```

I have tried out these routines on machines installed with AIX Versions 3.25, 4.2.1, and 4.3, without any problems. Also, your own definitions are not added to the snap output, so you don't need to worry too much about what data you put in there.

References:

- AIX man pages for the ODM commands (odmget).
- *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*.

Phil Pollard
Unix Systems Administrator (UK)

Code from *AIX Update* articles

Code from individual articles of *AIX Update* can be accessed on our Web site, at:

http://www.xephon.com/aixupdate.html

You will initially need the Web Registration Code shown on your address label; subsequently you will be asked to enter a word from the printed issue.

AIX 5L

AIX 5L is the culmination of an ambitious project by IBM to 'open-up' AIX while preserving the features that have seen it mature into an enterprise-strength operating system. This open approach includes Linux API compatibility, Unix System V printing support (optional on Power), and, of course, support of the Intel platform on the new Itanium 64-bit processors. Much of the openness is the result of Project Monterey, which set out to combine standard features of DYNIX/ptx and UnixWare with the proven strengths of AIX.

A number of hardware and software vendors have been running and validating AIX 5L on their hardware for quite some time. One of them, Bull, recently announced a successful test of AIX 5L on an 8-way Intel Itanium Server.

It's interesting to note that one of the cornerstone releases of AIX (3.2.5) was announced just 10 years ago. There is an interesting history of AIX from 3.2.5 to the present posted on IBM's Web site at http://www-1.ibm.com/servers/aix/pdf/S2S19Oct00.pdf.

ENHANCEMENT HIGHLIGHTS

JFS2

JSF2 characteristics include:

- Maximum (theoretical) file size is 4 petabytes (JFS is 64GB).
- Maximum File System size is 1 terabytes (JFS is 1 terabytes).
- Inode limit in JFS2 is dynamic and limited by disk size, JFS is fixed.
- Directory organization is B-tree (JFS is linear).

JFS2 will be the only filesystem available on Intel platforms. Both JFS and JFS2 will be available on Power Architecture.

Workload Manager enhancements

Workload Manager enhancements include:

- An API for external applications to modify system parameters.
- 270 controllable classes.
- Fully dynamic – all changes can be made with WLM running.
- A number of other enhancements.

Binary compatibility

Binary compatibility with previous releases of AIX Version 4.

LVM

LVM characteristics include:

- Hot spare disk capability within a volume group. If a disk cannot be restarted, then its mirror copy will be migrated to a hot spare disk if a disk of the proper size exists.
- Hot Spot management – currently there are no tools at the LVM level that will identify heavily-accessed partitions by the number of I/Os. There are other tools that could provide similar information like *iostat* or *filemon*, but they do not directly identify the partitions at the LVM level. Hot spot management provides two commands – one that will identify the hot spots, and another that migrates the hot spot to a different location.
- Variable LTG – today the LVM shipped with all versions of AIX 5L has a constant maximum transfer size of 128KB, which is known within LVM as the Logical Track Group (LTG). Enhancements are provided to allow a Volume Group (VG) LTG size to be specified at VG creation time and also to change the LTG size of an existing volume group. Different LTG sizes of 128KB, 256KB, 512KB, and 1024KB are now supported, and verification for the sizes of 512KB and 1024KB are underway. The different LTG sizes depend on the maximum transfer size of the member disks of the volume group.

RAS

RAS characteristics include:

- Scripts for basic analysis of dumps.
- Error storm management. For example, hung TTY port errors that scream error log entries can now be reduced to a few entries with counts.
- The ability to generate a core file in an application without termination.

Web serving

Web serving characteristics include:

- Performance enhancements to HTTP get engine.
- Java2 (Version 1.3) in base OS.

Other

Other features include:

- */proc* filesystem. The */proc* filesystem is a filesystem that provides access to the state of each process and thread in the system. The file system is mounted over */proc*. Standard system call interfaces such as *open()*, *read()*, *write()*, and *lseek()* are used to access */proc* files. Programs such debuggers can use */proc* to control a process that is being debugged. */proc* provides the ability to stop and start threads in a process, trace syscalls, trace signals, read and write virtual memory in a process, and other capabilities.
- Capability of printing entire books from documentation library.
- Data management API in JFS.

NEAR FUTURE ENHANCEMENTS

2001

In 2001 expect:

- NUMA system enabling.

- Power4 enabling with LPAR.
- Multi-server Web-based System Manager.
- Linux Application Environment.

2002

In 2002 expect:

- NUMA/SMP performance tuning.
- McKinley enabling for Itanium-based processors.
- RAS enhancements.
- Dynamic partitioning support.

OBTAINING A COPY OF AIX 5L INSTALLATION MEDIA AND DOCUMENTATION

The initial AIXL Version 5 release was due to ship sometime in December 2000. AIX 5L Version 5.1 is available now for Power as an Early Adopter Release (PRPQ), and Intel as a Developer Pre-release. The current estimate is that a general release of AIX 5.1 for both Power and Intel will be sometime in the second quarter of 2001. The pre-release format is intended primarily for software developers who need to certify and test both on the Power RS/6000 platform and the new Intel Itanium processors. The process to obtain the pre-release is in two steps:

1 Register at http://www.developer.ibm.com/sgi-bin/register your ID/password is displayed back to you right away.

2 Go to http://www.ibm.com/servers/aix/cdoffer/cdoffer.html to order the AIX 5L media and install the documentation.

FINAL COMMENT

If your environment is anything like mine, you are not waiting on the edge of your seat to be able to create a 1-terabyte filesystem (how many zeroes is that again?). However there are some very exciting features and enhancements coming our way in AIX 5L.

REFERENCES

Much of the material for this article came from a 'Webinar' (Web-based seminar) held on 7 November 2000. A copy of the Webinar presentation can be obtained at:

http://www-1.ibm.com/servers/aix/pdf/aix5/webinar07nov00.pdf.

Other Web areas for reference are AIX toolbox for Linux applications – http://www-1.ibm.com/servers/aix/products/aixos/linux/index.html – and AIX 5L Technical Preview White Paper – http://www-1.ibm.com/servers/aix/pdf/AIX50dr3ms.pdf.

David Miller
Database Architect
Baystate Health Systems (USA)

Hypertext Information Base and 'man'

WHAT IS 'MAN'?

Unix's **man** command is a portal to on-line reference information on commands, subroutines, and files, providing information on the specified topics. The Hypertext Information Base contains AIX documentation in HTML format, also containing information provided by the **man** command. The Hypertext Information Base is supplied on a CD labelled *Hypertext Library*.

HOW DOES MAN DISPLAY MANUAL PAGES?

The search path that the **man** command uses is a list of directories separated by colons (':'), the list being that of directories in which manual pages can be found. The default path is specified by the *MANPATH* environment variable.

The **man** command displays the manual pages as follows:

1 The **man** command searches **nroff**'s directories (*man1*, *man2*, etc), which are located in the directory */usr/share/man*.

2 **man** also searches the 'formatted version' directories (*cat1*, *cat2*, etc), which are also located in the directory */usr/share/man*. If the formatted version is available and has a modification time that is more recent than the **nroff** command's source, the **man** command displays it. Otherwise, the manual page is formatted using the **nroff** utility and displayed. If the user has the necessary authority, the formatted manual page is then deposited in its proper place so that later invocations of it do not require formatting.

   There is no **nroff** source for the supplied manual pages. However, you can put the source in the **man** directories so the **man** command can locate and process it.

3 If the **man** command does not find a manual page in either */usr/share/man/man* or */usr/share/man/cat*, it tries the Hypertext Information Base. This is in */usr/share/man/info* and contains the operating system's documentation. When reading the hypertext databases, the **man** command doesn't put formatted manual pages in the */usr/share/man/cat* directory. Instead, it strips formatting information from the manual page, wraps lines so they fit on the display, and displays the manual page using the command described by the *PAGER* environment variable.

STORING HIB ON THE HARD DISK

Download the contents of the hypertext CD-ROM to your hard disk as follows:

1 Unmount your CD-ROM filesystem using the following command:

```
umount /dev/cd0
```

2 Remove the CD-ROM filesystem using the command:

```
rmfs -r /hypercd
```

3 Create a mount point using the command:

```
mkdir /hypercd.tmp
```

4 Mount the temporary CD-ROM filesystem using the command:

```
mount -rv cdrfs /dev/cdØ /hypercd.tmp
```

5 You must have a filesystem mounted at */hypercd*. If the filesystem does not exist, enter the following commands:

```
rmdir /hypercd
SIZE=`df -k /hypercd.tmp|awk '/dev/ {print $2 * 1.2 }'`
crfs -v jfs -g rootvg -a size=$SIZE -m /hypercd -A yes -p rw
```

6 Mount the filesystem using the command:

```
mount /hypercd
```

7 You can now download the contents of this CD-ROM using the commands:

```
cd /hypercd.tmp
find . -print|cpio -pdmuv /hypercd
```

The contents of the CD-ROM are downloaded to your hard disk. Before using the CD-ROM for another task, don't forget to unmount it first using the command **umount /dev/cd0**. You may also remove the temporary mount point */hypercd.tmp*.

RUNNING MAN WITH FORMATTED PAGES ON YOUR HARD DISK

If you want to use the **man** command to display pre-formatted pages stored on your hard disk, make sure that the pages are in the */usr/man/cat?* directories.

The shell script below, **emp.sh** (extract man pages), extracts manual pages from the Hypertext Information Base on the CD-ROM and stores them in */usr/man/cat* directories. Once the script is run, the **man** command can use the pages to display relevant information.

Note the use of the continuation character, '➤', in the code below to indicate a formatting line break that does not appear in the original source code.

EMP.SH

```
#! /bin/ksh
################################################################################
```

```
# Name     : emp.sh ( extract man pages )
# Overview : This script extracts man pages from the Hypertext
#            Information Base CD-ROM and saves them as files
#            in the appropriate /usr/man/cat? directory.
# Notes    : 1. Usage: emp.sh <section>
#               where <section> is 1 for system commands
#                                  2 for system calls
#                                  8 for system maintenance documents
#            2. The script contains the following functions:
#               o InitializeVariables
#               o HandleInterrupt
#               o RootUser
#               o DisplayMessage
#               o ProcessExit
#               o ParseCommandLine
#               o ExtractManualPages
#               o ViewExceptionReport
#               o FormatUnderscores
#               o PrintFile
##############################################################################
# Name     : InitializeVariables
# Overview : Initializes all required variables.
##############################################################################
InitializeVariables ()
{
# command line arguments
P_SECTION=""
# define the root man directory
MAN_DIR="/usr/man/cat\${P_SECTION}"
# define location of man entries
# location of system commands
SYSTEM_COMM_DIR="/bin \
                 /usr/bin"
# location of system calls
SYSTEM_CALL_DIR="/lib"
# location of system maintenance commands
SYSTEM_MAINT_COMM_DIR="/usr/sbin"
# temporary files
TEMP_FILE="/tmp/pmp_$$.tmp"
TEMP_FILE_1="/tmp/pmp_$$_1.tmp"
TEMP_FILE_2="/tmp/pmp_$$_1.tmp"
EXCEPTION_REPORT="/tmp/pmp_$$_1.rep"
# define return codes
TRUE=Ø
FALSE=1
SEC=Ø
FEC=1
# define escape sequences
ESC="\ØØ33["
RVON="\ØØ33[7m"                 # reverse video on
```

```
RVOFF="\Ø033[27m"                # reverse video off
BOLDON="\Ø033[1m"                # bold on
BOLDOFF="\Ø033[22m"              # bold off
BON="\Ø033[5m"                   # blinking on
BOFF="\Ø033[25m"                 # blinking off
# define sleep duration
SLEEP_DURATION=3
# messages
OSERROR="\${ERR_MSG}${RVOFF}"
WORKING="Working........${RVOFF}"
INVALID_ARG_COUNT="Invalid number of arguments${RVOFF}"
USAGE="Usage:emp.sh \<section\>${RVOFF}"
INVALID_ARG_TYPE="Invalid argument${RVOFF}"
DUP_ARG="Duplicate argument${RVOFF}"
INVALID_SECTION="Invalid section specified\; value must be 1, 2, or
➤  8 ${RVOFF}"
SECTION_NOT_NUMERIC="Section must be numeric${RVOFF}"
NO_MAN_DIR="Directory \${MAN_DIR} does not exist${RVOFF}"
ROOT_ACCT_REQ="The script must be executed from root account${RVOFF}"
PRINT_OK="Successfully submitted print job${RVOFF}"
PRINT_NOT_OK="Failed to submit print job${RVOFF}"
# exit codes
SEC=Ø   # success exit code
FEC=99  # failure exit code
# return codes
TRUE=Ø
FALSE=1
ERROR="${RVON}emp.sh:ERROR:"
INFO="${RVON}emp.sh:INFO:"
# define signals
SIGHUP=1   ; export SIGHUP     # when the session is disconnected
SIGTERM=15 ; export SIGTERM    # kill command
SIGTSTP=18 ; export SIGTSTP    # ctrl-z command
}
#############################################################################
# Name     : HandleInterrupt
# Overview : The function calls ProcessExit.
#############################################################################
HandleInterrupt ()
{
WriteMessage I "${INTERRUPT}"
ProcessExit $FEC
}
#############################################################################
# Name     : MoveCursor
# Input    : Y and X coordinates
# Returns  : None
# Overview : Moves the cursor to the required location (Y, X).
#############################################################################
MoveCursor ( )
```

```
{
YCOR=$1
XCOR=$2
echo "${ESC}${YCOR};${XCOR}H"
}
##############################################################################
# Name     : DisplayMessage
# Overview : Displays a message.
# Input    : 1. Message type (E = Error, I = Information)
#            2. Error code as 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     : RootUser
# Overview : The function checks to see if the user is root or not.
# Returns  : TRUE if the user is root, FALSE otherwise.
##############################################################################
RootUser ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP
USER=`id | cut -d'(' -f2 | cut -d')' -f1`
if [ "${USER}" = "root" ]
then
    return $TRUE
else
    return $FALSE
fi
}
##############################################################################
# Name     : FormatUnderscores
# Overview : Assigns an appropriate number of underscore characters
#            to the variable UNDERSCORE that's 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 exception report (Y/N)?:\c"
    read REPLY
    case $REPLY in
        n|N ) return $TRUE ;;
        y|Y ) break ;;
          * ) : ;;
    esac
done
# get the printer name
while true
do
    clear
    echo "Enter the printer's 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     : ParseCommandLine
# Overview : Parses command line parameters.
#############################################################################
ParseCommandLine ()
{
# check argument count
if [ $ARGC -ne 1 ]
then
    DisplayMessage E "${INVALID_ARG_COUNT}" N
    DisplayMessage I "${USAGE}"
    return $FALSE
fi
# process arguments
P_SECTION="${ARGV}"
# valid argument? (1, 2, or 8)
if ( [ `expr $P_SECTION + 0` -eq $P_SECTION ] ) > /dev/null 2>&1
then
    if [ $P_SECTION -ne 1 -a $P_SECTION -ne 2 -a $P_SECTION -ne 8 ]
    then
        DisplayMessage E "${INVALID_SECTION}"
        return $FALSE
    fi
else
    DisplayMessage E "${SECTION_NOT_NUMERIC}"
    return $FALSE
fi
}
#############################################################################
# Name     : ExtractManualPages
# Overview : Prepares pages man command for required section.
#############################################################################
ExtractManualPages ()
{
# initialize exception report
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
HEADER="Exception report for section ${P_SECTION} on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo " ${HEADER}  " > ${EXCEPTION_REPORT}
echo " ${UNDERSCORE}  " >> ${EXCEPTION_REPORT}
# evaluate manual directory
MAN_DIR=`eval echo ${MAN_DIR}`
# check existence of $MAN_DIR directory
if [ ! -d ${MAN_DIR} ]
```

```
then
    DisplayMessage E "${NO_MAN_DIR}"
    return $FALSE
fi
DisplayMessage I ${WORKING}
# process section 1
if [ $P_SECTION -eq 1 ]
then
    for DIR in ${SYSTEM_COMM_DIR}
    do
        for COMMAND in `ls "${DIR}"`
        do
          if [ -x "${DIR}/${COMMAND}" ]
          then
               # command is executable - extract pages from the
               # Hypertext Information Base.
               if ( man "${P_SECTION}" "${COMMAND}" 1> ${TEMP_FILE} \
                    2> ${TEMP_FILE_1} )
               then
                   MAN_FILE="${MAN_DIR}/${COMMAND}.${P_SECTION}"
                   mv ${TEMP_FILE} ${MAN_FILE}
               else
                   # write the exception record
                   cat ${TEMP_FILE_1} >> ${EXCEPTION_REPORT}
               fi
          fi
        done
    done
elif [ $P_SECTION -eq 2 ]
then
    egrep '^[a-z]' ${SYSTEM_CALL_DIR}/syscalls.exp | \
          awk {'print $1'} > ${TEMP_FILE}
    cat ${TEMP_FILE} | while read SYSTEMCALL
    do
        if ( man "${P_SECTION}" "${SYSTEMCALL}" 1> ${TEMP_FILE_1} \
             2> ${TEMP_FILE_2} )
        then
            MAN_FILE="${MAN_DIR}/${SYSTEMCALL}.${P_SECTION}"
            mv ${TEMP_FILE_1} ${MAN_FILE}
        else
            # write the exception record
            cat ${TEMP_FILE_2} >> ${EXCEPTION_REPORT}
        fi
    done
elif [ $P_SECTION -eq 8 ]
then
    CURDIR=`pwd`
    cd  ${SYSTEM_MAINT_COMM_DIR}
    file * | grep executable | awk -F: '{print $1}' > ${TEMP_FILE}
    cat ${TEMP_FILE} | while read SYSMCOMM
```

```
    do
        if ( man "${P_SECTION}" "${SYSMCOMM}" 1> ${TEMP_FILE_1} \
                                              2> ${TEMP_FILE_2} )
        then
            MAN_FILE="${MAN_DIR}/${SYSMCOMM}.${P_SECTION}"
            mv ${TEMP_FILE_1} ${MAN_FILE}
        else
            # write the exception record
            cat ${TEMP_FILE_2} >> ${EXCEPTION_REPORT}
        fi
    done
fi
cd $CURDIR
return $TRUE
}
################################################################################
# Name     : ViewExceptionReport
# Overview : Allows the user to view and print the exception report.
################################################################################
ViewExceptionReport ()
{
# check for records in exception record
if grep "There is not an entry" ${EXCEPTION_REPORT} > /dev/null 2>&1
then
    view ${EXCEPTION_REPORT}
    PrintFile "${EXCEPTION_REPORT}"
fi
return $TRUE
}
################################################################################
# Name     : ProcessExit
# Overview : Makes a graceful exit.
# Input    : Exit Code
################################################################################
ProcessExit ()
{
# assign parameter
EXIT_CODE="$1"
# remove temporary files
rm -f ${TEMP_FILE}
rm -f ${TEMP_FILE_1}
rm -f ${TEMP_FILE_2}
rm -f ${EXCEPTION_REPORT}
clear
exit $EXIT_CODE
}
################################################################################
# Name     : main
# Overview : Invokes all other functions.
################################################################################
```

```
main ()
{
InitializeVariables
trap "HandleInterrupt " $SIGTERM $SIGHUP $SIGTSTP
if ! RootUser
then
    DisplayMessage E "${ROOT_ACCT_REQ}"
    ProcessExit $FEC
fi
if ! ParseCommandLine
then
    ProcessExit $FEC
fi
if ! ExtractManualPages
then
    ProcessExit $FEC
fi
ViewExceptionReport
ProcessExit $SEC
}
# assign argument count
ARGC="$#"
# assign argument values
ARGV="$@"
# invoke main
main
```

VIEWING AIX DOCUMENTATION USING A BROWSER

1 Locate the shell script **rc.hyper** in directory */hpercd/bin*.

2 Start the custom http daemon using the command:

```
/hpercd/bin/rc.hyper
```

3 Start the browser and point to the following URL:

```
http://<ip_address_of_the_server>:7328
```

RC.HYPER

```
cd /hyperlib/bin
case $1 in
    stop) PID=`cat /hyperlib/data/logs/httpd.pid`; kill -1 $PID ;;
       *) ./httpd -d ../data >/dev/console 2>&1 ;;
esac
```

NOTES

1 The custom **httpd** (http listener) is started with the directory directive (**-d**) pointing to directory */hyperlib/data*, where **httpd** expects the following subdirectories:

   – *conf*

   – *logs*.

2 The value of *DocumentRoot* in the *httpd.conf* file in the *conf* subdirectory should be set to */hyperlib/lib*.

3 The first HTML page that is displayed is *TopNav.htm*, which is in the */hyperlib/lib* directory. Links on this page may have to be changed so that the browser has the correct URLs.

4 The directory */hyperlib/lib* should include the following subdirectories, which contain OS-related documentation:

   – *bookloc_en_US*

   – *bookloc_de_DE*

   – *bookloc_es_ES*

   – *bookloc_fr_fr*

   – *bookloc_IT_it*.

   The URLs in *TopNav.htm* must point to pages in these directories.

5 While the **httpd** is using the CD's filesystem, the filesystem cannot be unmounted.

CUSTOM MANUAL PAGES

cmp.sh (custom manual pages) is a shell script that allows the user to build a manual page for in-house commands or utilities. Once the script is run to customize a command's page, running **man <*command*>** displays the page.

CMP.SH

```
#! /bin/ksh
```

```
#############################################################################
# Name     : cmp.sh (customize man pages)
#
# Overview : This shell script allows the user to customize man
#            pages for in-house utilities.
#
# Notes    1. Usage: cmp.sh
#
#          2. The script contains the following functions:
#             o InitializeVariables
#             o HandleInterrupt
#             o MoveCursor
#             o RootUser
#             o DisplayMessage
#             o ProcessExit
#             o GetCommandName
#             o PrepareTemplate
#             o EditTemplate
#             o StoreEditedTemplate
#             o TestNewPage
#             o main
#          3. The script stores the man page in the /usr/man/cat1
#             directory, which must already exist.
#############################################################################
# Name     : InitializeVariables
# Overview : Initializes all required variables
#############################################################################
InitializeVariables ()
{
# define man directory for commands
MAN_DIR="/usr/man/cat1"
# temporary files
TEMP_FILE="/tmp/pmp_$$.tmp"
TEMPLATE_FILE="/tmp/cmp_$$_1.template"
# define return codes
TRUE=Ø
FALSE=1
SEC=Ø
FEC=1
# define escape sequences
ESC="\ØØ33["
RVON="\ØØ33[7m"                  # reverse video on
RVOFF="\ØØ33[27m"                # reverse video off
BOLDON="\ØØ33[1m"                # bold on
BOLDOFF="\ØØ33[22m"              # bold off
BON="\ØØ33[5m"                   # blinking on
BOFF="\ØØ33[25m"                 # blinking off
# define sleep duration
SLEEP_DURATION=3           # used by DisplayMessage ()
# messages
```

```
OSERROR="\${ERR_MSG}"
INTERRUPT="Program interrupted\; quitting early${RVOFF}"
WORKING="Working........${RVOFF}"
EDIT_TEMPLATE="Edit the man page template as required${RVOFF}"
VIEW_MAN_PAGE="Viewing new man page for command, \${COMMAND} ${RVOFF}"
USAGE="Usage:emp.sh \<section\>${RVOFF}"
INVALID_ENTRY="Invalid entry${RVOFF}"
NO_MAN_DIR="Directory \${MAN_DIR} does not exist${RVOFF}"
ROOT_ACCT_REQ="The script must be executed from root account${RVOFF}"
PRINT_OK="Successfully submitted print job${RVOFF}"
PRINT_NOT_OK="Failed to submit print job${RVOFF}"
# exit codes
SEC=Ø   # success exit code
FEC=99  # failure exit code
# return codes
TRUE=Ø
FALSE=1
ERROR="${RVON}cmp.sh:ERROR:"
INFO="${RVON}cmp.sh:INFO:"
# define signals
SIGHUP=2    ;  export SIGHUP     # when the session is disconnected
SIGTERM=15  ;  export SIGTERM    # kill command
SIGTSTP=18  ;  export SIGTSTP    # ctrl-z command
}
################################################################################
# Name     : HandleInterrupt
# Overview : The function calls ProcessExit.
################################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit $FEC
}
################################################################################
# Name     : MoveCursor
# Overview : Moves the cursor to the required location (Y, X).
# Input    : Y and X coordinates
################################################################################
MoveCursor  ( )
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP
YCOR=$1
XCOR=$2
echo ${ESC}${YCOR};${XCOR}H"
}
################################################################################
# Name     : DisplayMessage
# Overview : Displays a message.
# Input    : 1. Message type (E = Error, I = Informative)
#            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     : RootUser
# Overview : The function checks whether the user is root.
# Returns  : TRUE if user is root, FALSE otherwise.
##############################################################################
RootUser ()
{
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP
USER=`id | cut -d'(' -f2 | cut -d')' -f1`
if [ "${USER}" = "root" ]
then
    return $TRUE
else
    return $FALSE
fi
}
##############################################################################
# Name     : PrepareTemplate
# Overview : The function prepares the template for man pages.
##############################################################################
PrepareTemplate ()
{
echo "
Purpose


<     >

Syntax


<     >

Description
```

```
<     >

Bugs


<     >

See Also


<     >

" > ${TEMPLATE_FILE}
}
#############################################################################
# Name     : GetCommandName
# Overview : Allows the user to input the command name for the man
#            page.
# Returns  : $TRUE if a command name is supplied
#            $FALSE otherwise.
#############################################################################
GetCommandName ()
{
# get the command name
while true
do
    clear
    echo "Enter the command name for man page (q to quit):\c"
    read COMMAND
    case $COMMAND in
        "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
        q|Q) return $FALSE ;;
        *  ) return $TRUE ;;
    esac
done
}
#############################################################################
# Name     : EditTemplate
# Overview : Allows the user to edit the template for man pages.
#############################################################################
EditTemplate ()
{
DisplayMessage I "${EDIT_TEMPLATE}"
vi ${TEMPLATE_FILE}
}
#############################################################################
# Name     : StoreEditedTemplate
# Overview : Stores the edited template.
#############################################################################
```

```
StoreEditedTemplate ()
{
# format the template file
echo "                   ${COMMAND} command \n\n" > ${TEMP_FILE}
cat  ${TEMPLATE_FILE} >> ${TEMP_FILE}
# put a copy id man directory
if [ ! -d ${MAN_DIR} ]
then
    DisplayMessage E "${NO_MAN_DIR}"
    return $FALSE
fi
cp ${TEMP_FILE} ${MAN_DIR}/${COMMAND}.1
return $TRUE
}
################################################################################
# Name     : TestNewPage
# Overview : Allows the user to view and edit the new man page.
# Returns  : $TRUE if editing is not required
#            $FALSE if editing is required
################################################################################
TestNewPage ()
{
DisplayMessage I "${VIEW_MAN_PAGE}"
man ${COMMAND}
while true
do
    clear
    echo "Do you wish to edit the man page (Y/N)?:\c"
    read REPLY
    case $REPLY in
        y|Y ) return $FALSE ;;
        n|N ) return $TRUE ;;
          * ) DisplayMessage E "${INVALID_ENTRY}" ;;
    esac
done
}
################################################################################
# Name     : ProcessExit
# Overview : The function makes a graceful exit.
# Input    : Exit code
################################################################################
ProcessExit ()
{
# assign parameter
EXIT_CODE="$1"
# remove temporary files
rm -f ${TEMP_FILE}
rm -f ${TEMPLATE_FILE}
clear
exit $EXIT_CODE
```

```
}
##############################################################################
# Name     : main
# Overview : Invokes all other functions.
# Notes    1. The function calls the following functions:
#             o InitializeVariables
#             o RootUser
#             o DisplayMessage
#             o ProcessExit
#             o PrepareTemplate
#             o GetCommandName
#             o EditTemplate
#             o StoreEditedTemplate
#             o TestNewPage
##############################################################################
main ()
{
InitializeVariables
trap "HandleInterrupt " $SIGTERM $SIGHUP $SIGTSTP
if ! RootUser
then
    DisplayMessage E "${ROOT_ACCT_REQ}"
    ProcessExit $FEC
fi
PrepareTemplate
if ! GetCommandName
then
    ProcessExit $FEC
fi
while true
do
    EditTemplate
    if ! StoreEditedTemplate
    then
        ProcessExit $FEC
    fi
    if TestNewPage
    then
        break
    fi
done
ProcessExit $SEC
}
# invoke main
main
```

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

Keeping your system updated using Inventory Scout and Microcode Discovery Service

Maintenance of a typical RS/6000 machine involves periodic updates of the AIX operating system and other software. This is a structured process, which is facilitated by PTFs issued by IBM to fix various bugs. From time to time these PTFs are bundled into maintenance levels, which can be ordered from local IBM representatives on CDs. This article describes tools used to keep up with updates to system firmware in RS/6000 machines. Firmware is software that is imbedded in various hardware devices from which a computer system is built – the main system processor, the service processor, I/O and communication adapters, the hard disks and tapes drives.

Please note that the Inventory Scout tool is supported only on a PowerPC processor-based computer; older machines, built using the Power processors, are not supported.

INSTALLATION OF INVENTORY SCOUT

Inventory Scout is a tool that surveys servers and workstations for hardware and software information. Inventory Scout can be installed on AIX Versions 4.1.5+, 4.2, and 4.3.

The package consists of three separate filesets, which can be downloaded from the following URL:

http://techsupport.services.ibm.com/rs6k/invscout/invreadme.html.

At the time of writing of this article, the filesets where last updated on 15 December 2000 and had the following versions:

Package Function	*Package Name*	*Package Version*
Runtime Executable	Invscut.rte	1.1.0.1
Logical Database	Invscout.ldb	1.1.0.0
Logical Database Update	Invscout.ldb_1.1.0.6.bff	1.1.0.6

The packages should be installed on each RS/6000 machine in your organization. The following steps have to be performed in order to install the software:

1 Download the packages to a temporary directory such as */tmp/IS*.

2 Log in as root user.

3 Enter the command *smitty install_latest*.

4 Type */tmp/IS* in the field titled *INPUT device/directory* and press *Enter*.

5 Press the F4 key in the field entitled *SOFTWARE to install* and select the Inventory Scout packages; press *Enter* to confirm your selection.

6 Press *Enter* again in order to start the installation.

After the completion of the installation, verify its success by running the following command:

```
lslpp -l '*inventory*'.
```

CONFIGURATION OF INVENTORY SCOUT

Inventory Scout is implemented in two different ways – a command line-based utility named **invscout**, and a daemon named **invscoutd**.

The **invscout** command can be used to produce a survey of a host's Vital Product Data (VPD) or a host's microcode survey. Simply run the command **invscout** and after its completion the file */var/adm/invscut/hostname.mup* will contain a Microcode Survey Upload File that can be uploaded to a Web server for processing. Once on a Web server, a CGI script will compare the file's data with the database and produce a formatted Web page, reporting information on the status of the microcode and URLs of sites to obtain the latest versions of supported microcode. The file */var/adm/invscut/invs.mrp* will contain the formatted text report file. The **invscout** has to be invoked with flag **-v** in order to produce the VPD survey upload file */var/adm/invscout/hostname.vup*. This file can be uploaded to a Web server for processing. Once on the server, a CGI forwards the file to the repository of server configurations and produces a Web page indicating the status of the operation. Please note that in the name of upload files' *hostname* is replaced by the name of the actual host on which the command is running.

You can use the daemon version of the tool in order to avoid manual downloading of inventory files. First you should set a password for **invscout** user, which is created during the install. To start a daemon, log in as **root** and type the following command:

```
/usr/sbin/invcoutd  /var/adm/invscout/invscoutd.log
```

To enable automatic start-up of the daemon after the reboot of the computer, place the following line into the */etc/initab:*

```
invscoutd:2:once:/usr/sbin/invscutd /var/adm/invscout/invscoutd.log
```

To verify that the daemon is running, type:

```
ps -ef|grep invscoutd
```

USING THE MICROCODE DISCOVERY SERVICE

The IBM Microcode Discovery Service is accessed through the following URL:

http://techsupport.services.ibm.com/rs6k/mds.html.

You need a secure Internet connection and a PC or Unix workstation running Netscape's Navigator or Microsoft's Internet Explorer browser. The service is implemented as a Java applet, so execution of Java applets downloaded from the Internet should be enabled in the browser's options.

The access to service can be performed in two ways. If your computer is running the **invscoutd** daemon, you can supply the applet with the hostname of the machine, the password of the invscout user, and the port to communicate with the daemon (808 by default). Alternatively, you can download the file to a different applet. In both cases the data discovered by the Inventory Scout is transferred to the remote IBM computer.

The transferred information is encrypted using the SSL protocol and contains only the data that is needed to determine the need to upgrade microcode of various system devices – computer type, list of devices with current level of microcode, level of the operating system, names and levels of installed operating system filesets.

The Microcode Discovery Service Report, produced after the analysis of your computer's data, is done by the IBM server, and contains two major sections – System View and Microcode Package View.

The System View contains general information about the analysed computer including:

1 Host name.

2 Machine Model (the output of **uname -M** command).

3 Machine ID (the ninth and tenth characters of the output from the **uname -m** command).

4 The table of system devices that have imbedded microcode (this table appears with more details under the Microcode Package View heading).

5 The table that lists the names and description of all computer devices that don't contain microcode, identifiable by the tool.

The Microcode Package View contains detailed information about the microcode used by the analysed computer. It is formatted as a table with the following headings:

1 Device/system description – the formal name of the device.

2 Package name – the formal name of the Microcode Package for the device.

3 README – a hyperlink to the README file, containing detailed information about the microcode, including installation instructions.

4 Latest available – a hyperlink pointing to a file containing the latest version of the device's microcode.

5 Release date – the date that the microcode was made available.

6 Need to update:

   1 **Yes** indicates that the installed version is downlevel and needs to be updated.

   2 **No** indicates that the installed version is the latest available and no action is required.

3 **Unknown** indicates that Inventory Scout cannot determine the installed version. Follow instructions in the README file to determine the installed version.

7 Impact:

1 **Availability** indicates fixes that improve the availability of resources.

2 **Data** indicates fixes that resolve a customer data error.

3 **Function** indicates fixes that add (introduce) or affect system/machine operation with regard to features, connectivity, and resources.

4 **Security** indicates fixes that improve or resolve security issues.

5 **Serviceability** indicates fixes that influence problem determination/fault isolation and maintenance with regard to diagnostic errors, incorrect FRU calls, false error (no operational impact).

6 **Performance** indicates fixes that improve or resolve throughput or response times.

7 **Usability** indicates fixes that improve user interfaces/messages.

8 Severity:

1 **HIPER** (High Impact/Pervasive) indicates that the fix should be installed as soon as possible.

2 **SPE** (Special Attention) indicates that the fix should be installed at the earliest convenience. These fixes resolve low probability but high impact problems.

3 **ATT** (Attention) indicates that a fix should be installed at the earliest convenience. These fixes resolve low probability, low to medium impact problems.

4 **PE** (Programming Error) indicates that the fix can be installed when convenient. These fixes resolve minor problems.

If the value under the 'Need to update?' heading of the report contains

indication **Unknown**, you can perform the following actions in order to determine the microcode level of the device:

1 If the device is a hard disk or tape drive:

   1 Enter **lscfg -vl hdisk*** for a hard disk or **lscfg -vl rmt*** for a tape drive.

   2 Find the drive in question.

   3 If the field for the drive contains a **ROS** or **Device Specific.(Z1)** or **Loadable** level field, the drive most likely has uploadable microcode.

2 If the device is an adapter:

   1 Enter **lscfg -vp**.

   2 Find the adapter in question.

   3 If the field for the adapter contains a **Loadable** or **ROS** level, the adapter most likely has uploadable microcode.

In order to manually check the availability of updated microcode, access the following URL:

http://www.rs6000.ibm.com/support/micro/download.html.

MICROCODE DOWNLOAD AND INSTALLATION PROCEDURE

Download the appropriate format of the microcode by clicking on hyperlink under the 'Latest Available' heading in the Microcode Package View report. Unzip the microcode file. To unzip the file on an AIX system, enter the following commands:

```
chmod +x filename.bin
./filename.bin
```

You will be asked to supply a password, which is ****RS/6000****.

The archive contained in the **filename.bin** will unpack itself into a number of files, possibly located in a new subdirectory. Read and follow the instructions in the README file that comes with the microcode. In some cases the README file will provide detailed instructions on installing the microcode. In other cases the README

file will point to the Description file found on the Web page for detailed instructions.

You have to remember that installation of most microcode packages will require a system reboot or a temporary interruption in the usage of the affected device. Additionally you should not power off the system or the affected device during the microcode download because this may damage the device, requiring it to be replaced.

REFERENCE

1 Inventory Scout man pages: invscout(1), invscoutd(1)

2 Inventory Scout Users Guide

Aleksander Polyak
System Engineer
APS (Israel)

vi extended

INTRODUCTION

It would be nice if, every time we use **vi** to edit a file, it could save the current version of the file using a version number (in other words, age the file). It would also be nice if we could specify which version of the file we want to edit. **vie** is a utility that extends the functionality of **vi** in those areas.

The utility components are:

1 The standard **vi** editor.

2 The shell script **agefile.sh**, which ages a file.

3 The main shell script **vie.sh**.

The definitions used are:

- original_file = <file name>

- specific_version = <file name>.version_no
- latest_version = the original file
- new_file = a file that we create.
- version_no starts with ‘1’.

PROCESSING OVERVIEW

The pseudo-code below outlines the logic of the program:

```
IF vie is invoked with a specific version of the file
THEN
    IF the file exists
    THEN
        MAKE a copy of the file
        INVOKE vi with this file
        IF the file has changed
        THEN
            AGE the file by invoking agefile.sh with this file name
            MAKE this copy the latest version
        ELSE
            EXIT
        END IF
    ELSE
        ISSUE error message
    END IF
ELSIF vie is invoked with the original file
THEN
    MAKE a copy of the file
    INVOKE vi with this file
    IF the file has changed
    THEN
        AGE the file by invoking agefile.sh with this file name
    ELSE
        EXIT
    END IF
ELSIF vie is invoked with a new file
THEN
    INVOKE vi with this file
END IF
```

VIE.SH

```
#############################################################################
# Name     : vie.sh (vi extended)
# Overview : This script extends the functionality of the vi editor by
#            ageing the file that is being edited.
```

```
# Notes    : 1 The methods of invoking the script are as follows:
#               - vie.sh
#               - vie.sh <file name>
#               - vie.sh <file name> <version number>
#             2 The script implements the following rules:
#               - When vie.sh is invoked with no parameters, it invokes
#                 the vi editor with no file name.
#               - When vie.sh in invoked with only a file name, it
#                 invokes the vi editor with that file name and ages the
#                 file, if the file exists and has been changed.
#               - When vie.sh in invoked with a file name and a specific
#                 version number, it invokes the vi editor only if the
#                 version of the file exists. If the file has changed, it
#                 ages the file and makes edited version the latest or
#                 current version.
#             3 The script calls the following programs:
#               - agefile.sh
#               - vi
#             4 The script contains the following functions:
#               - InitializeVariables
#               - PrepareFileName
#               - DisplayMessage
#               - MoveCursor
#               - ProcessExit
#               - ParseCommandLine
#               - AgeFile
#               - main
################################################################################
# Name     : InitializeVariables
# Overview : The function initializes all variables.
################################################################################
InitializeVariables ()
{
P_FILE = ""              # command line argument, file name
P_VERSION_TO_EDIT = ""   # command line argument, version number to edit
FILE_TO_EDIT = ""        # file name to invoke vi editor with
SCRIPT_PATH = /usr/bin   # location for agefile.sh  and  vi
FILE_TO_EDIT_BEFORE = "/tmp/vie_$$.before" # before copying file to edit
FILE_TO_EDIT_AFTER = "/tmp/vie_$$.after"   # after copying file to edit
# checksum and bytes details of edited file and
# original file
CHKSUM_BEFORE = ""
CHKSUM_AFTER = ""
CHKSUM_BYTES_BEFORE = ""
CHKSUM_BYTES_AFTER = ""
# new file flag
NEW_FILE = ""
# editing current file flag
EDIT_LATEST = ""
# exit codes
SEC = Ø   # success exit code
```

```
FEC = 99  # failure exit code
# return codes
TRUE = Ø
FALSE = 1
# escape sequences
ESC = "\ØØ33["
RVON = "\ØØ33[7m"           # reverse video on
RVOFF = "\ØØ33[27m"         # reverse video off
BOLDON = "\ØØ33[1m"         # bold on
BOLDOFF = "\ØØ33[22m"       # bold off
BON = "\Ø033[5m"            # blinking on
BOFF = "\Ø033[25m"          # blinking off
SLEEP_DURATION = 4          # seconds allowed for sleep command
ERROR = "vie.sh:ERROR:"
INFO = "vie.sh:INFO:"
# message
WORKING = "Working"
INTERRUPT = "Program interrupted - quitting early"
INVALID_ARGC = "Invalid argument count"
DUP_ARG = "\${ARG_TYPE}, is a duplicate argument type"
OSERROR = "\${ERR_MSG}"
USAGE = "Usage: agefile.sh \<full file name\> \<version to edit\>"
NOT_NUMERIC = "Version count must be numeric"
INVALID_VERSION_NO = "Version count must be greater than Ø"
DUP_ARG = "Duplicate argument"
FILE_NOT_FOUND = "File \${FILE_TO_EDIT} is not found"
# define signals
SIGHUP = 1    ;  export SIGHUP     # session disconnected
SIGINT = 2    ;  export SIGINT     # ctrl-c
SIGTERM = 15  ;  export SIGTERM    # kill command
SIGTSTP = 18  ;  export SIGTSTP    # interactive stop (ctrl-z)
}
################################################################################
# Name     : ParseCommandLine
# Overview : The function parses the command line arguments.
# Returns  : TRUE or FALSE
# Notes    : 1. The first argument is assigned to P_FILE
#               The second argument is assigned to P_VERSION_TO_EDIT
################################################################################
ParseCommandLine ()
{
# check argument count
if [$ARGC -gt 2]
then
    DisplayMessage E "${INVALID_ARGC}" "N"
    DisplayMessage I "${USAGE}"  N
    return $FALSE
elif [$ARGC -eq Ø]
then
    return $TRUE
fi
```

```
# process argument line $ARGL
# extract file name
P_FILE = `echo "${ARGL}" | cut -d' ' -f1`
# extract version to edit
if [$ARGC -ne 2]
then
    return $TRUE
fi
P_VERSION_TO_EDIT = `echo "${ARGL}" | cut -d' ' -f2`
if ["${P_VERSION_TO_EDIT}" ! = ""]
then
    if ! ValidateVersionNo "${P_VERSION_TO_EDIT}"
    then
        return $FALSE
    fi
fi
return $TRUE
}
################################################################################
# Name     : ProcessExit
# Overview : The function implements a graceful exit.
# Input    : Exit code
################################################################################
ProcessExit ()
{
EXIT_CODE = "$1"
rm -f ${FILE_TO_EDIT_BEFORE}
rm -f ${FILE_TO_EDIT_AFTER}
clear
exit ${EXIT_CODE}
}
################################################################################
# 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 : The function displays the incoming message.
# Input    : 1. Message type (E = Error, I = Information)
#            2. Error code
#            3. Message to be acknowledged flag (Y = yes, N = no)
# Notes    : 1. If the message is to be acknowledged, the functions
#               displays the message and waits for an input as a signal
#               of an acknowledgement. If the message is not to be
```

```
#               acknowledged, the function appends three periods ('...')
#               to the end of the message to be displayed to signify
#               that this message does not need to be acknowledged.
##############################################################################
DisplayMessage ()
{
MESSAGE_TYPE = $1
MESSAGE_TEXT = "`eval echo $2`"
ACKNOWLEDGE_FLAG = "$3"
# default the message acknowledge flag
if ["${ACKNOWLEDGE_FLAG}" = ""]
then
    ACKNOWLEDGE_FLAG = "Y"
fi
#
MoveCursor 24 1
if ["${MESSAGE_TYPE}" = "E"]
then
    if ["${ACKNOWLEDGE_FLAG}" = "N"]
    then
        echo "`eval echo ${RVON}${ERROR}`${MESSAGE_TEXT}...${RVOFF}\c"
    else
        echo "`eval echo ${RVON}${ERROR}`${MESSAGE_TEXT}${RVOFF}\c"
    fi
else
    if ["${ACKNOWLEDGE_FLAG}" = "N"]
    then
        echo "`eval echo ${RVON}${INFO}`${MESSAGE_TEXT}...${RVOFF}\c"
    else
        echo "`eval echo ${RVON}${INFO}`${MESSAGE_TEXT}${RVOFF}\c"
    fi
fi
# examine message acknowledge flag
if ["${ACKNOWLEDGE_FLAG}" = "Y"]
then
    read DUMMY
else
    sleep ${SLEEP_DURATION}
fi
return ${TRUE}
}
##############################################################################
# Name     : ValidateVersionNo
# Overview : The function validates the version of the file.
# Input    : The file name
# Returns  : FALSE if not numeric or less than 1
#            TRUE otherwise
##############################################################################
ValidateVersionNo ()
{
# assign parameter
```

```
P_VERSION = "$1"
if ([`expr $P_VERSION + 0` -eq $P_VERSION] ) > /dev/null 2>&1
then
    if [$P_VERSION -gt 0]
    then
        return $TRUE
    else
        DisplayMessage E "${INVALID_VERSION_NO}" N
        return $FALSE
    fi
else
    DisplayMessage E "${NOT_NUMERIC}" N
    return $FALSE
fi
}
################################################################################
# Name     : PrepareFileName
# Overview : The function processes the name of the file that's to be
#            passed to vi. It also sets the $NEW_FILE and $EDIT_LATEST
#            flags.
# Notes    : 1. The function assigns the file name to $FILE_TO_EDIT .
################################################################################
PrepareFileName ()
{
# check the parameter
if ["${P_FILE}" = ""]
then
    # no file name specified
    NEW_FILE = "Y"
    FILE_TO_EDIT = ""
    return $TRUE
fi
# check the existence of file
if [! -f "${P_FILE}"]
then
    # new file
    NEW_FILE = "Y"
    FILE_TO_EDIT = "${P_FILE}"
    return $TRUE
fi
# check which version is being edited
if ["${P_VERSION_TO_EDIT}" = ""]
then
    # the latest or current version being edited
    EDIT_LATEST = Y
    FILE_TO_EDIT = "${P_FILE}"
    cp ${FILE_TO_EDIT} ${FILE_TO_EDIT_BEFORE}
    return $TRUE
else
    # specific version being edited
    EDIT_LATEST = N
```

```
    FILE_TO_EDIT = "${P_FILE}.${P_VERSION_TO_EDIT}"
    if [! -f "${FILE_TO_EDIT}"]
    then
        DisplayMessage E "${FILE_NOT_FOUND}" N
        return $FALSE
    fi
    EDIT_LATEST = N
    cp ${FILE_TO_EDIT} ${FILE_TO_EDIT_BEFORE}
    return $TRUE
fi
}
################################################################################
# Name     : AgeFile
# Overview : The function ages the file that is to be edited by calling
#            the agefile.sh script.
################################################################################
AgeFile ()
{
# check for new file flag
if ["${NEW_FILE}" = "Y"]
then
    # no ageing required
    return $TRUE
fi
# check whether the file has changed
CHKSUM_BEFORE = `cksum ${FILE_TO_EDIT_BEFORE} | awk {'print $1'}`
CHKSUM_BYTES_BEFORE = `cksum ${FILE_TO_EDIT_BEFORE} | awk {'print $2'}`
CHKSUM_AFTER = `cksum ${FILE_TO_EDIT} | awk {'print $1'}`
CHKSUM_BYTES_AFTER = `cksum ${FILE_TO_EDIT} | awk {'print $2'}`
if [$CHKSUM_BEFORE -eq $CHKSUM_AFTER]  && \
   [$CHKSUM_BYTES_BEFORE -eq $CHKSUM_BYTES_AFTER]
then
    # nothing has changed
    return $TRUE
fi
# which version has been edited
if ["${EDIT_LATEST}" = "Y"]
then
    # latest file has been edited - save the edited version
    cp ${FILE_TO_EDIT}  ${FILE_TO_EDIT_AFTER}
    # restore the 'before' image
    cp ${FILE_TO_EDIT_BEFORE}  ${FILE_TO_EDIT}
    # age the file
    ${SCRIPT_PATH}/agefile.sh ${P_FILE}
    # restore the 'after' image
    cp ${FILE_TO_EDIT_AFTER} ${FILE_TO_EDIT}
else
    # specific version has been edited - save edited version
    cp ${FILE_TO_EDIT}  ${FILE_TO_EDIT_AFTER}
    # restore the 'before' image
    cp ${FILE_TO_EDIT_BEFORE}  ${FILE_TO_EDIT}
```

```
    # age the file
    ${SCRIPT_PATH}/agefile.sh ${P_FILE}
    # make the edited version the latest one
    cp ${FILE_TO_EDIT_AFTER} ${P_FILE}
fi
return $TRUE
}
##############################################################################
# Name     : main
# Overview : The function implements the processing structure.
# Notes    : 1. The function calls the following function:
#                - ParseCommadLine
#                - PrepareFileName
#                - AgeFile
#                - ProcessExit
##############################################################################
main ()
{
if ! ParseCommandLine
then
    ProcessExit $FEC
fi
if ! PrepareFileName
then
    ProcessExit $FEC
fi
# invoke vi
${SCRIPT_PATH}/vi ${FILE_TO_EDIT}
# age file(s)
AgeFile
ProcessExit $SEC
}
InitializeVariables
# package the command line
ARGC = $#
ARGL = "$@"
# invoke main
main
```

AGEFILE.SH

Note that this script can be run on its own to age an existing file and purge the unwanted versions of the file.

```
##############################################################################
# Name     : agefile.sh (age file)
# Overview : The script ages a specific file to a later version and
#             purges the unwanted  versions.
# Notes    : 1. The command is invoked as follows:
#                 agefile.sh <file name> <version>
```

```
#                 where <file name> = the name of the file that
#                                     requires ageing (mandatory)
#                       <version>   = the number of versions of the file
#                                     to keep (optional)
#               Example:
#               If a directory contains the following files:
#                 /tmp/f1.dat
#                 /tmp/f1.dat.1
#                 /tmp/f1.dat.2
#                 /tmp/f1.dat.3
#               then the command:
#                 agefile /tmp/f1.dat
#               is equivalent to the following:
#                 copy  f1.dat.3  f1.dat.4
#                 copy  f1.dat.2  f1.dat.3
#                 copy  f1.dat.1  f1.dat.2
#                 copy  f1.dat    f1.dat.1
#               and the command:
#                 agefile   /tmp/f1.dat  2
#               is equivalent to the following:
#                 copy    f1.dat.1  f1.dat.2
#                 copy    f1.dat    f1.dat.1
#                 remove  f1.dat.3
#             2. When specifying the file name, the absolute path name
#                must be used unless the file is in the current
#                directory.
#             3. If the number of versions to keep is not specified, the
#                script just ages the file to a later version.
#             4. Parameters must be passed from the command line.
#             5. Specify the file name without the version number.
#             6. The script contains the following functions:
#                - InitializeVariables
#                - ParseCommandLine
#                - DisplayMessage
#                - MoveCursor
#                - AgeFile
#                - HandleInterrupt
#                - ProcessExit
##############################################################################
# Name     : InitializeVariables
# Overview : The function initializes all the variables.
##############################################################################
InitializeVariables ()
{
# command line parameter holders
P_FILE = ""
P_VERSIONS_TO_KEEP = ""
# temporary file
TEMP_FILE = "/tmp/agefile_$$.tmp"
# error file
ERROR_FILE = "/tmp/agefile_$$.err"
```

```
# exit codes
SEC = Ø   # success exit code
FEC = 99  # failure exit code
# return codes
TRUE = Ø
FALSE = 1
# escape sequences
ESC = "\ØØ33["
RVON = [7m                  # reverse video on
RVOFF = [27m                # reverse video off
BOLDON = [1m                # bold on
BOLDOFF = [22m              # bold off
BON = [5m                   # blinking on
BOFF = [25m                 # blinking off
SLEEP_DURATION = 4          # seconds for sleep command
ERROR = "agefile.sh:ERROR:"
INFO = "agefile.sh:INFO:"
# messages
WORKING = "Working"
INTERRUPT = "Program interrupted - quitting"
INVALID_ARGC = "Invalid argument count"
DUP_ARG = "\${ARG_TYPE} is a duplicate argument type"
INVALID_ENTRY = "Invalid entry"
OSERROR = "\${ERR_MSG}"
USAGE = "Usage: agefile.sh \<full file name\> \<versions to keep\>"
FILE_NOT_EXIST = "File, \${P_FILE} does not exist"
INVALID_FILE = "File, \${FILE_NAME} is invalid"
FILE_NOT_READABLE = "File, \${P_FILE_NAME} is not readable by user"
FILE_NOT_WRITEABLE = "File, \${P_FILE_NAME} is not writeable by user"
INVALID_VERSION_NO = "Invalid version count"
DUP_ARG = "Duplicate argument"
FILE_NOT_PURGED = "Failed to remove file \${FILE_NAME}"
FILE_NOT_COPIED = "Failed to copy file \${FILE_NAME}"
FILE_NAME_MISSING = "Must provide a file name"
# define signals
SIGHUP = 1    ;  export SIGHUP     # when session disconnected
SIGINT = 2    ;  export SIGINT     # ctrl-c
SIGTERM = 15  ;  export SIGTERM    # kill  command
SIGTSTP = 18  ;  export SIGTSTP    # interactive stop (ctrl-z)
}
################################################################################
# Name     : HandleInterrupt
# Overview : The function calls ProcessExit.
################################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}" N
ProcessExit $FEC
}
################################################################################
# Name     : MoveCursor
```

```
# Input    : Y and X coordinates
# Returns  : None
# Overview : Moves the cursor to the required location (Y, X).
##############################################################################
MoveCursor  ()
{
YCOR = $1
XCOR = $2
echo "${ESC}${YCOR};${XCOR}H"
}
##############################################################################
# Name     : DisplayMessage
# Overview : The function displays the incoming message.
# Input    : 1. Message type (E = Error, I = Information)
#            2. Error code
#            3. Message to be acknowledged flag (Y = yes, N = no)
# Notes    : 1. If the message is to be acknowledged, the functions
#               displays the message and waits for an input as a signal
#               of an acknowledgement. If the message is not to be
#               acknowledged, the function appends three periods ('...')
#               to the end of the message to be displayed to signify
#               that this message does not need to be acknowledged.
##############################################################################
DisplayMessage ()
{
MESSAGE_TYPE = $1
MESSAGE_TEXT = "`eval echo $2`"
ACKNOWLEDGE_FLAG = "$3"
# default message acknowledge flag
if ["${ACKNOWLEDGE_FLAG}" = ""]
then
    ACKNOWLEDGE_FLAG = "Y"
fi
MoveCursor 24 1
if ["${MESSAGE_TYPE}" = "E"]
then
    if ["${ACKNOWLEDGE_FLAG}" = "N"]
    then
        echo "`eval echo ${RVON}${ERROR}`${MESSAGE_TEXT}...${RVOFF}\c"
    else
        echo "`eval echo ${RVON}${ERROR}`${MESSAGE_TEXT}${RVOFF}\c"
    fi
else
    if ["${ACKNOWLEDGE_FLAG}" = "N"]
    then
        echo "`eval echo ${RVON}${INFO}`${MESSAGE_TEXT}...${RVOFF}\c"
    else
        echo "`eval echo ${RVON}${INFO}`${MESSAGE_TEXT}${RVOFF}\c"
    fi
fi
# examine message acknowledge flag
```

```
if ["${ACKNOWLEDGE_FLAG}" = "Y"]
then
    read DUMMY
else
    sleep ${SLEEP_DURATION}
fi
return ${TRUE}
}
################################################################################
# Name     : ValidateFileName
# Overview : The function validates a specific file.
# Input    : File name
# Returns  : FALSE if the file does not exist or cannot be read nor
#                  written.
#            TRUE otherwise.
################################################################################
ValidateFileName ()
{
# assign parameter
P_FILE_NAME = "$1"
if [! -f "${P_FILE_NAME}"]
then
    return $FALSE
elif [! -r "${P_FILE_NAME}"]
then
    DisplayMessage E "${FILE_NOT_READABLE}" N
    return $FALSE
fi
}
################################################################################
# Name     : ValidateVersionNo
# Overview : The function validates the parameter that specifies the
#            number of versions of the file to keep.
# Input    : File name
# Returns  : FALSE if either the input isn't a number or is less than
#                  one.
#            TRUE otherwise.
################################################################################
ValidateVersionNo ()
{
# assign parameter
P_VERSIONS = "$1"
if ([`expr $P_VERSIONS + 0` -eq $P_VERSIONS]) > /dev/null 2>&1
then
    if [$P_VERSIONS -gt 0]
    then
        return $TRUE
    else
        return $FALSE
    fi
else
```

```
    return $FALSE
fi
}
##############################################################################
# Name     : AgeFile
# Overview : The function ages a specific file to a later version and
#            then purges unwanted versions of the file, if required.
# Returns  : TRUE if successful
#            FALSE otherwise
# Notes    : 1. The function switches to the directory where the file to
#               be aged resides.
##############################################################################
AgeFile ()
{
DisplayMessage I "${WORKING}" N
# does first version of the program exist ?
FIRST_VERSION = "${P_FILE}.1"
if [! -f ${FIRST_VERSION}]
then
  # first version does not exist - copy the original to version number 1
    FILE_NAME = "${P_FILE}"
    (cp ${P_FILE} ${FIRST_VERSION}) > ${ERROR_FILE} 2>&1
    if [$? -ne Ø]
    then
        DisplayMessage E "${FILE_NOT_COPIED}" N
        ERR_MSG = `cat ${ERROR_FILE}`
        DisplayMessage E "${OSERROR}" N
        return $FALSE
    else
        return $TRUE
    fi
fi
# versions of the file exist, so work out which is the oldest
VERSION_COUNT = 1
while true
do
    THIS_VERSION = "${P_FILE}.${VERSION_COUNT}"
    if [-f ${THIS_VERSION}]
    then
        VERSION_COUNT = `expr $VERSION_COUNT + 1`
        continue
    else
        # oldest version found
        break
    fi
done
# start ageing the file from bottom up
# eg file.1Ø (oldest file) becomes file.11, etc
while [$VERSION_COUNT -ge 1]
do
    if [$VERSION_COUNT -eq 1]
```

```
    then
        # age the original to version 1
        FILE_NAME = "${P_FILE}"
        (cp ${P_FILE} ${P_FILE}.${VERSION_COUNT}) > ${ERROR_FILE} 2>&1
        if [$? -ne Ø]
        then
            DisplayMessage E "${FILE_NOT_COPIED}"  N
            ERR_MSG = `cat ${ERROR_FILE}`
            DisplayMessage E "${OSERROR}" N
            return $FALSE
        else
            break
        fi
    fi
    # age the previous version to this version
    THIS_VERSION = "${P_FILE}.${VERSION_COUNT}"
    PREVIOUS_VERSION_COUNT = `expr $VERSION_COUNT - 1`
    PREVIOUS_VERSION = "${P_FILE}.${PREVIOUS_VERSION_COUNT}"
    FILE_NAME = "${PREVIOUS_VERSION}"
    (cp $PREVIOUS_VERSION $THIS_VERSION) > ${ERROR_FILE} 2>&1
    if [$? -ne Ø]
    then
        DisplayMessage E "${FILE_NOT_COPIED}" N
        ERR_MSG = `cat ${ERROR_FILE}`
        DisplayMessage E "${OSERROR}" N
        return $FALSE
    fi
    # decrement version count
    VERSION_COUNT = `expr $VERSION_COUNT - 1`
done
# purge the file if requested
if ["${P_VERSIONS_TO_KEEP}" = ""]
then
    # no purges requested
    return $TRUE
fi
PURGING_VERSION_NO = `expr $P_VERSIONS_TO_KEEP + 1`
while true
do
    VERSION_TO_PURGE = ${P_FILE}.${PURGING_VERSION_NO}
    if [-f ${VERSION_TO_PURGE}]
    then
        # this version qualifies for purge
        FILE_NAME = "${VERSION_TO_PURGE}"
        (rm ${VERSION_TO_PURGE}) > ${ERROR_FILE} 2>&1
        if [$? -ne Ø]
        then
        # failed to remove the file
        ERR_MSG = `cat ${ERROR_FILE}`
        DisplayMessage E "${FILE_NOT_PURGED}" N
        DisplayMessage E "${OSERROR}" N
```

```
        return $FALSE
        fi
    else
        # all versions are required
        return $TRUE
    fi
    PURGING_VERSION_NO = `expr $PURGING_VERSION_NO + 1`
done
return $TRUE
}
################################################################################
# Name     : ParseCommandLine
# Overview : The function parses the command line arguments.
# Returns  : TRUE or FALSE
# Notes    : 1. The first argument is assigned to P_FILE
#               The second argument is assigned to P_VERSIONS_TO_KEEP
################################################################################
ParseCommandLine ()
{
# check the argument count
if [$ARGC -gt 2]
then
    DisplayMessage E "${INVALID_ARGC}" "N"
    DisplayMessage I "${USAGE}" N
    return $FALSE
elif [$ARGC -eq Ø]
then
    DisplayMessage E "${INVALID_ARGC}" "N"
    DisplayMessage I "${USAGE}" N
    return $FALSE
fi
# process argument line $ARGL - extract file name
P_FILE = `echo "${ARGL}" | cut -d' ' -f1`
if ["${P_FILE}" ! = ""]
then
    if ! ValidateFileName "${P_FILE}"
    then
        FILE_NAME = "${P_FILE}"
        DisplayMessage E "${INVALID_FILE}" N ;
        return $FALSE
    fi
fi
# extract versions to keep
if [$ARGC -ne 2]
then
    return $TRUE
fi
P_VERSIONS_TO_KEEP = `echo "${ARGL}" | cut -d' ' -f2`
if ["${P_VERSIONS_TO_KEEP}" ! = ""]
then
    if ! ValidateVersionNo "${P_VERSIONS_TO_KEEP}"
    then
```



```
        DisplayMessage E "${INVALID_VERSION_NO}" N
        return $FALSE
    fi
fi
return $TRUE
}
##############################################################################
# Name     : ProcessExit
# Overview : The function implements a graceful exit.
# Input    : Exit code
##############################################################################
ProcessExit ()
{
EXIT_CODE = "$1"
rm -f ${ERROR_FILE}
clear
exit ${EXIT_CODE}
}
##############################################################################
# Name     : main
# Overview : This function implements the processing structure. If
#            parameters are not supplied in the command line, the
#            function attempts to obtain them interactively.
# Notes    : 1. The function calls the following functions:
#               - InitializeVariables
#               - ParseCommandLine
#               - GetParameters
#               - AgeFile
#               - ProcessExit
##############################################################################
main ()
{
InitializeVariables
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# parse command line
if ! ParseCommandLine
then
    ProcessExit $FEC
fi
AgeFile
ProcessExit $SEC
}
# package command line arguments and values
ARGC = $#
ARGL = "$@"
# invoke main
main
```

AIX news

McAfee has begun shipping Version 5.0 of its GroupShield anti-virus suite for Lotus Domino, now with added protection for AIX servers.

It's five times faster than previous GroupShield versions and now comes with Outbreak Manager, Content Scanning, and ePolicy Orchestrator technology, promising enhanced real-time on-demand scanning for viruses and malicious code, including Lotus scripts and button bombs.

Working with McAfee's ePolicy Orchestrator technology, the tool allows administrators to selectively enable, filter, and prioritize system alerts and direct them to e-mail, pagers, or other devices.

For further information contact:
Network Associates, 3965 Freedom Circle, Santa Clara, CA 95054, USA.
Tel: (408) 988 3832.
URL: http://www.mcafeeb2b.com.

* * *

Trend Micro has begun shipping Version 2.5 of its ScanMail virus protection software for Lotus Notes, with enhanced performance, virus scanning and cleaning, message handling, and integration features.

Supporting Lotus Domino on AIX, NT, OS/390, and OS/400, and Solaris, it uses real-time e-mail virus-scanning features so that viruses can be detected and cleaned from incoming messages before they have a chance to infect the desktop.

Specific improvements include policy-based configuration and improved management by embedding ScanMail into the Domino R5 Administration Console. Also, ScanMail for Notes is now integrated into Trend's cross-platform central management console, the Trend Virus Control System.

For further information contact:
Trend Micro, 10101 N DeAnza Blvd, #400 Cupertino, CA 95014, USA.
Tel: (408) 257 1500.
URL: http://www.antivirus.com/products/smln/.

* * *

Blaze Software has released Version 3.2 of its Blaze Advisor Solutions Suite, which contains new features, allowing designers to create complete rule maintenance applications. With the resultant applications, non-technical users can maintain business policies and personalization rules for production systems without, claims the company, requiring any help from their IT department.

The technology allows users to control the business rules that define how a business operates in any particular situation, separating business logic from screen navigation, GUI controls, and database interactions.

Advisor's supported platforms include AIX, OS/400, OS/390, Compaq NSK, Solaris, and Windows 2000 and NT.

For further information contact:
Brokat Americas, 6625 The Corners Parkway, Suite 500, Norcross, GA 30092, USA.
Tel: (678) 533 4600.
URL: http://www.brokat.com/int/press/2001/advisor.html.