64

AIX update

February 2001

In this issue

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.

Editor s
Trevor Eddolls and Richard Watson

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.

Managing cron jobs under HACMP

I have written the Change_cron.ksh script to allow application cron jobs to be controlled in a High Availability Cluster Multiprocessing (HACMP) environment. This is useful because it gives the administrator the flexibility to run certain application cron jobs only if the application has been made active on a node by HACMP.

The script accepts three parameters – the user, either ‘enable’ or ‘disable’, and the TAG ID. The user is the user whose crontab file you wish to manipulate. The enable or disable options either enable or disable the cron jobs, which are marked with the TAG ID. The TAG ID is an identifier that is at the end of each crontab entry. It is used to identify which crontab entries are to be manipulated by the script. This allows you to have other cron jobs in the same user crontab file, which are not manipulated by the script, or cron jobs for different HACMP resource groups.

The first thing you need to do is update your crontab entries and ensure that they all end with the relevant TAG ID. The crontab file needs to exist on all nodes that are participating in the HACMP cluster. This format is exactly the same as a normal crontab entry, but it must end with ‘# TAG’, where TAG is an application-unique ID. Below is an example of an Oracle crontab entry:

```
# Oracle crontab file
0 * * * * /usr/local/bin/check_database sid # Checks
```

Now, on all the systems within the HACMP cluster you need to disable the cron jobs by running the following command (please note that, when the script disables the cron jobs, it will place two hash symbols (##) at the start of each line in the crontab file):

```
/usr/local/bin/Change_cron.ksh oracle disable Checks
```

Now you need to ensure that the crontab entries are disabled after a system reboot. This is to ensure that all cron jobs are disabled at reboot time, because, if the system had crashed, the cron jobs may still be enabled. To do this, add an entry to the /etc/inittab file with the following command:

```
mkitab "changecron:2:once:/usr/local/bin/Change_cron.ksh oracle disable
Checks"
```

Now you need to add the following line to your HACMP application start script:

```
/usr/local/bin/Change_cron.ksh oracle enable Checks
```

Now you need to add the following line to your HACMP application stop script:

```
/usr/local/bin/Change_cron.ksh oracle disable Checks
```

That's it. Now test that all is working as expected:

```
#!/usr/bin/ksh
# Script: Change_cron.ksh
# Author: Steve Diwell - Jedi Technology Ltd.
# Aim: To enable or disable APPLICATION cron entries under HACMP control
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin
[[ "$VERBOSE_LOGGING" = "high" ]] && set -x
[[ $# != 3 ]] && {
     echo "Usage:" $0 "User [enable/disable] Tag"
     exit 1
     }
# Assign the input parameters.
USER=$1
FUNC=$2
TAG=$3
# Does the user have a crontab file?
[[ ! -f /var/spool/cron/crontabs/${USER} ]] && {
     echo "Sorry, the user \"${USER}\" does not have a crontab file."
     exit 1
     }
# Setup the cron variables and temporary work files.
CRON_MASTER=/var/spool/cron/crontabs/${USER}
CRON_BACKUP=${CRON_MASTER}.backup
CRON_OUT=/tmp/Change_cron_${USER}.$$
# Enable cron jobs.
if [[ ${FUNC} = "enable" ]]
  then
     if grep -q "^##.*# ${TAG}" ${CRON_MASTER}
       then
# To remove ## from start of lines ending ${TAG}
          sed "s;\(^##\)\(.*\)\(# ${TAG}\);\2\3;" ${CRON_MASTER} >
${CRON_OUT}
       else
          exit 0        # Already enabled.
     fi
fi
```

```
# Disable cron jobs.
if [[ ${FUNC} = "disable" ]]
  then
     if grep -q "^##.*# ${TAG}" ${CRON_MASTER}
       then
          exit 0        # Already disabled.
       else
# To add the ## to start of lines ending ${TAG}
          sed "s;\(.*\)\(# ${TAG}\);##\1\2;" ${CRON_MASTER} >
${CRON_OUT}
     fi
fi
# Back-up the users crontab file.
cp -p ${CRON_MASTER} ${CRON_BACKUP}
[[ $? != 0 ]] && {
     echo "Error creating the ${USER} crontab file backup!!"
     exit 1
     }
# Now create the new crontab for the user.
su ${USER} -c /usr/bin/crontab ${CRON_OUT} 1>/dev/null
[[ $? != 0 ]] && {
     echo "Error creating the ${USER} new crontab file!!"
     exit 1
     }
sleep 1
# Remove the temporary work file.
rm -f ${CRON_OUT}
exit 0
```

Steve Diwell
Senior Consultant
Jedi Technology Ltd (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 need the user-id shown on your address label.

AIX admin helpers

In the busy life of an AIX administrator, especially one who is responsible for a large number of systems and has a minimal staff, one can use all the help one can get. I have included several very helpful utilities that can be used to quickly diagnose some common problems an admin group might face:

- **Findbig** – in seconds, identify the (n) largest files in directory tree (n).
- **Biggestinroot** – quickly locate the largest files in the '/' root filesystem that are *not* in filesystems.
- **Check100.sh** – a watcher, which runs from crontab every 15 minutes to detect when one of your systems may be running exceptionally hard, monitoring CPU and I/O. It will send mail notification detailing the possible performance issue observed.
- **Hicpu** – quickly display the current 15 most CPU-intensive processes on the system.
- **DF** – a better version of the AIX *df* command, with everything lined up nicely.

FINDBIG

When you notice that one or more filesystems are running out of room, it can be a tedious process to locate the files that are consuming so much of the available space. This is often the case where one or more very large files are the culprits. For example, a job may be logging thousands of messages to a log file that grows and grows until it bottoms out the filesystem. But that log file may be lurking in some subdirectory and you don't know where to look. Meanwhile the 100% filled filesystem is causing other system problems, so you have no time to start hunting around with the *ls* command. To quickly find the largest files in a filesystem or directory tree, findbig can save a lot of time.

It has two optional flags:

- The **-R** flag indicates that it should look down the directory tree recursively.
- The **–N** flag can be used to specify the number of files to display (the default is 20).

Examples:

- **findbig -N10 /tmp** (would display in numeric reverse order, the largest 10 files in /tmp).
- **findbig –R /var** (the largest 20 files in /var filesystem including subdirectories).
- **findbig –R –N5 /UserData/lib** (largest five files in the tree /UserData/lib incl. subdirectories).

```
#!/bin/ksh
#
# findbig
#
# Will display 20 (default) biggest files sorted in descending
# size order.
#
# Specifying the -R argument makes it recursive down through
# the tree.
# Specifying the -N argument used for requesting more or less
# than the default number of files to list, which is 20.
#
# Examples:
#    To find the 20 biggest files in the filesystem /UserData:
#         findbig /UserData
#    To find the 32 biggest files in /var and all its subdirectories:
#         findbig -R -N32 /var
#
#----------------------------------------------------------------------
recursive="false"

while getopts :RN: name
do
    case $name in
    R)      recursive="true";;
    N)      filestolist="$OPTARG";;
    ?)      printf "Usage: %s [-R] [-N value] starting-directory\n" $0
            exit 2;;
    esac
done
```

```
shift $((OPTIND -1))
dirspec=$*
echo "starting directory is: ${dirspec}"
echo "Working..."

if [[ "${recursive}" = "true" ]]
then
   find ${dirspec:=/} -ls | sort +6nr | head -${filestolist:=20}
else
   find ${dirspec:=/}* -prune -ls |sort +6nr |head -${filestolist:=20}
fi
#
```

BIGGESTINROOT

The difference between this script and findbig is that when the '/' root filesystem gets filled to 100%, this is a special circumstance, and it can be very difficult to locate the file(s) that have caused the problem. The reason is that the directories under '/' may be simply that, directories, or they may be mount points for filesystems. So a utility is needed to distinguish between the two. The biggestinroot script will only consider directories under '/' that are *not* filesystem mount points. Filesystem mountpoints must be ignored, since their files are obviously not what is filling up the '/' filesystem. This script will display the 10 largest files in each non-filesystem directory to the screen, and output the list to a file as well, for saving or printing.

```
#!/bin/ksh
#
# biggestinroot
#
# Show the biggest 10 files in each non-filesystem root directory, ie
# only in the root filesystem (/).
#
#-----------------------------------------------------------------------

[[ -f /tmp/bigfiles.lis ]] && rm /tmp/bigfiles.lis

print "\n10 Biggest files in the non-filesystem root directories" \
>  /tmp/bigfiles.lis

print "======================================================"   \
>> /tmp/bigfiles.lis

print "Report date: $(date)\n\n" >> /tmp/bigfiles.lis
print "WORKING..."

#...For each directory under the / filesystem, determine whether or not
#   it is the mount-point for a filesystem, or simply a directory under
#   the root filesystem. This can be determined by checking if it is in
#   /etc/filesystems.

for dirnam in $(ls /)
do
    if [ -d /${dirnam} ]
    then
     grep -q "/${dirnam}:" /etc/filesystems && continue

     print "Processing directory: ${dirnam}"
     print "\n<<<<Directory: ${dirnam}>>>>\n" >> /tmp/bigfiles.lis

     findbig -R /${dirnam} >> /tmp/bigfiles.lis  #findbig does rest
    fi
done

print "Press enter to page '/tmp/bigfiles.lis' to the screen:\c"
read response extra
pg /tmp/bigfiles.lis

#eoj
```

CHECK.SH

When you have a lot of systems to keep an eye on, it's impossible to constantly check on how all of them are running. To help notify you when there may be a performance issue which requires your attention, running the check100.sh from cron can be an extra pair of eyes. We run it between the hours of 6am and 7pm, every 15 minutes. It samples the CPU and I/O information from *vmstat* over a ten-minute period. If during that time the average CPU and/or I/O appear abnormally high, it sends a mail message detailing what it found. After examining the mail message, you can decide whether the condition it points out is unexpected or not, and take the appropriate action as necessary.

Here is a sample line to add to your root crontab file:

```
0,15,30,45 6-19 * * * ksh /MBSYSMGR/check100.sh >
/SYSLOGS/check100.log 2>&1
```

Code

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

```
# check1ØØ.sh
# Michael Stanton
# Description:
# This script will use vmstat sampling to determine whether the system
# is running hard enough to make the system admin aware of a possible
# performance issue. Five samples are taken, with two minutes between
# each sample. So the analysis is done over a period of ten minutes.
# During this time, if the average system CPU or I/O are greater than
# the expected norm, the system admin will be notified via e-mail, and
# the mail will contain a snapshot of the busiest cpu processes running
# at the time.
#
# Scheduling this script:
# It is recommended that this script be run from the root crontab like
# so:
#
# Ø,15,3Ø,45 6-19 * * * ksh /MBSYSMGR/check1ØØ.sh >
#                                             SYSLOGS/check1ØØ.log 2>&1
#
# This will run it every day between the hours of 6am and 7pm, every 15
# minutes.
#
# The directories for the script (/MBSYSMGR) and the log (/SYSLOGS) can
# be set to your preference.
#
#-----------------------------------------------------------------------

#...mail subroutines
SEND_MAIL () {

trap '' ERR
    msgoptions=""
    case $1 in
   "NOTIFY_CPU")
        echo " $(cat /tmp/check1ØØ.out)"                > $MAIL_FILE
        echo " Message Source: $CURR_SHELL             " >> $MAIL_FILE
        echo "                                         " >> $MAIL_FILE
        MAIL_SUBJECT="${place} : CPU running ${cpu_avg}%."
        ;;

   "NOTIFY_IO")
        echo " $(cat /tmp/check1ØØ.out)"                > $MAIL_FILE
        echo " Message Source: $CURR_SHELL             " >> $MAIL_FILE
        echo "                                         " >> $MAIL_FILE
        MAIL_SUBJECT="${place} : Very high IO-wait ${io_avg}%."
        ;;

   "NOTIFY_OVERALL")
        echo " $(cat /tmp/check1ØØ.out)"                > $MAIL_FILE
        echo " Message Source: $CURR_SHELL             " >> $MAIL_FILE
        echo "                                          " >> $MAIL_FILE
        MAIL_SUBJECT="${place} : High combined CPU (${cpu_avg}%) & I-O
Wait (${io_avg}%)"
        ;;

    * )
        echo "                                          " >  $MAIL_FILE
        echo "Invalid parm passed to SEND_MAIL function." >> $MAIL_FILE
        echo "                                          " >> $MAIL_FILE
        echo "Please investigate and correct...         " >> $MAIL_FILE
        echo "                                          " >> $MAIL_FILE
        echo "Message Source: $CURR_SHELL               " >> $MAIL_FILE
        MAIL_SUBJECT="${place} : INVALID SEND_MAIL PARAMETER"
        ;;
    esac

#...substitute the desired user or distribution list for AIXTECHS_DIS..

    mail -s "${MAIL_SUBJECT}" "AIXTECHS_DIS" <  ${MAIL_FILE}
    return

}
# **********************************************************************
# Start of main shell body
 **********************************************************************
. /etc/profile

MAIL_FILE="/tmp/check100.bdy"
CURR_SHELL="$(basename $0)"
place=$(hostname)
#
echo "check100.sh procedure beginning at $(date)..."
#
# Make 5 passes at 2-minute intervals to test the user-mode,
# kernel-mode and IO-wait times using vmstat.
pass1=""
pass2=""
pass3=""
pass4=""
pass5=""
#
echo "Displaying the topcpu processes..." > /tmp/check100.out

pass1=$(vmstat 1 2 | tail -1)

cpu_u_1=$(echo ${pass1} |awk '{print $14 }')
echo "cpu_u_1: $cpu_u_1"
cpu_k_1=$(echo ${pass1} |awk '{print $15 }')
echo "cpu_k_1: $cpu_k_1"
io_1=$(echo ${pass1}    |awk '{print $17 }')
```

```
sleep 120

pass2=$(vmstat 1 2 | tail -1)

cpu_u_2=$(echo ${pass2} |awk '{print $14 }')
echo "cpu_u_2: $cpu_u_2"
cpu_k_2=$(echo ${pass2} |awk '{print $15 }')
echo "cpu_k_2: $cpu_k_2"
io_2=$(echo ${pass2}    |awk '{print $17 }')
sleep 120

pass3=$(vmstat 1 2 | tail -1)

cpu_u_3=$(echo ${pass3} |awk '{print $14 }')
echo "cpu_u_3: $cpu_u_3"
cpu_k_3=$(echo ${pass3} |awk '{print $15 }')
echo "cpu_k_3: $cpu_k_3"
io_3=$(echo ${pass3}    |awk '{print $17 }')
sleep 120

pass4=$(vmstat 1 2 | tail -1)

cpu_u_4=$(echo ${pass4} |awk '{print $14 }')
echo "cpu_u_4: $cpu_u_4"
cpu_k_4=$(echo ${pass4} |awk '{print $15 }')
echo "cpu_k_4: $cpu_k_4"
io_4=$(echo ${pass4}    |awk '{print $17 }')
sleep 120

pass5=$(vmstat 1 2 | tail -1)

cpu_u_5=$(echo ${pass5} |awk '{print $14 }')
echo "cpu_u_5: $cpu_u_5"
cpu_k_5=$(echo ${pass5} |awk '{print $15 }')
echo "cpu_k_5: $cpu_k_5"
io_5=$(echo ${pass5}    |awk '{print $17 }')

#
typeset -i cpu_u_total=0
typeset -i cpu_k_total=0
typeset -i io_total=0
typeset -i cpu_total=0
typeset -i io_total=0
typeset -i cpu_avg=0
typeset -i io_avg=0
typeset -i overall_avg=0

cpu_u_total=$((${cpu_u_1}+${cpu_u_2}+${cpu_u_3}+${cpu_u_4}+${cpu_u_5}))
cpu_k_total=$((${cpu_k_1}+${cpu_k_2}+${cpu_k_3}+${cpu_k_4}+${cpu_k_5}))
io_total=$((${io_1}+${io_2}+${io_3}+${io_4}+${io_5}))
#
```

```
echo "cpu_u_total is: $cpu_u_total" >> /tmp/check100.out
echo "cpu_k_total is: $cpu_k_total" >> /tmp/check100.out
echo "io_total is: $io_total"       >> /tmp/check100.out

cpu_total=$((${cpu_u_total} + ${cpu_k_total}))
#
echo "cpu_total is: $cpu_total" >> /tmp/check100.out

cpu_avg=$((${cpu_total}/5))
io_avg=$((${io_total}/5))
overall_avg=$((${cpu_avg} + ${io_avg}))

echo "The cpu_avg was: ${cpu_avg}."         >> /tmp/check100.out
echo "The io_avg was: ${io_avg}."           >> /tmp/check100.out
echo "The overall_avg was: ${overall_avg}." >> /tmp/check100.out
#
#
# The hicpu script should be called from your admin scripts directory. #
We use a directory called /MBSYSMGR. Hicpu will display the busiest
# CPU processes, and include this in the mail message body.
#
if [ ${cpu_avg} -gt 98 ]
then
    /MBSYSMGR/hicpu >> /tmp/check100.out
    SEND_MAIL NOTIFY_CPU
elif [ ${io_avg} -gt 70 ]
then
    /MBSYSMGR/hicpu >> /tmp/check100.out
    SEND_MAIL NOTIFY_IO
elif [ ${overall_avg} -gt 99 ]
then
    /MBSYSMGR/hicpu >> /tmp/check100.out
    SEND_MAIL NOTIFY_OVERALL
fi
#
[[ -f /tmp/check100.out ]] && rm /tmp/check100.out  #delete tempfile
[[ -f /tmp/check100.bdy ]] && rm /tmp/check100.bdy  #delete mailtemp
#
# eoj
```

HICPU

Based upon the ‘C’ column of the AIX *ps –ef* command, this utility immediately identifies those top 15 processes that are currently compute-bound and are hammering the CPU. It sorts them in numeric reverse order, so the topmost process is the one running the most intensively. The ‘C’ column represents a counter that is maintained by the kernel for the number of consecutive times the process was found to be in the CPU run-queue in line for getting CPU-time. The higher the number, the more relentless the process.

```
#!/bin/ksh
#
# hicpu
# Michael Stanton
#
# Description:
# This one-line script will print the top 15 busiest CPU processes.
# It is based upon the 'C' column of the ps output. The higher this
# value, the more intense the process is running on the CPU at the
# time.
#
# As stated in the man pages for ps :
#
# C (-f, l, and -l flags) CPU utilization of process or thread,
#incremented each time the system clock ticks and the process or thread
#is found to be running. The value is decayed by the scheduler by
#dividing it by 2 once per second. For the sched_other policy, CPU
#utilization is used in determining process scheduling priority. Large
#values indicate a CPU intensive process and result in lower process
#priority whereas small values indicate an I/O intensive process and
#result in a more favorable priority.
#----------------------------------------------------------------------
#
# We strip out the Header and our own PID ($$) as well before reporting
# on the top 15 processes. Then we sort in reverse numeric order, with
# the most intense CPU process at the top of the list.
#
ps -ef | egrep -v "STIME|$$" | sort +3 -r | head -n 15
#
#eoj
```

DF

This utility makes the output from *df –kv* look so much nicer and easier to read. It makes the columns line up and I never use the normal *df* command anymore since I made it my habit to type *DF* instead. For long filesystem names, the skewed appearance is no more – just neat columns rearranged for readability.

```
#!/bin/ksh
#
# DF
#
# Michael Stanton
```

```
#
# Description:
#
# Rearranging of df -kv output so that the columns all line up neatly.
#
# Note: output may be sorted as desired. e.g. "DF | sort -k4" would
#    display the output sorted by the %used field.
#
#----------------------------------------------------------------------
#
df -kv |awk '{ print $2,$3,$4,$5,$9,$1 }' \
     | while read num1 num2 num3 str1 str2 str3
       do
        printf "%9s %9s %9s  %5s    %-14s  %s\n" \
$num1 $num2 $num3 $str1 $str2 $str3
       done
#
#eoj
```

In a future article I will present more AIX admin helpers. I hope these utilities prove as useful to you in your environment as they have been to me in mine.

Mike Stanton
Supervisor, Mid-range Systems
Mercedes-Benz (USA)

The Apache Web server – part 2

This month's instalment concludes this article on an Apache administration script.

```
########################################################################
# Name     : ShowHttpdProcessDetails
# Overview : The function displays the http listener processes.
# Notes    : 1. The function calls the following functions:
#              - PrintFile
#              - DisplayMessage
#              - FormatUnderscores
########################################################################
ShowHttpdProcessDetails ()
{
# are there any instances running?
if ps -eaf | grep "httpd" | grep -v "grep" > /dev/null 2>&1
```

```
then
    # http listener running
    :
else
    DisplayMessage E "${LISTENER_NOT_RUNNING}" N
    return $FALSE
fi
# initialize temporary file
DATETIME = `date "+%d/%m/%Y at %H:%M:%S"`
HEADER = "Listener Process Details on ${DATETIME}"
FormatUnderscores "${HEADER}"
echo "      ${HEADER}" > ${TEMP_FILE_1}
echo "      ${UNDERSCORE}\n" >> ${TEMP_FILE_1}
ps -eaf | grep "httpd" | grep -v "grep" >> ${TEMP_FILE_1}
view ${TEMP_FILE_1}
PrintFile "${TEMP_FILE_1}"
}
########################################################################
# Name     : RestartHttpdImmediately
# Overview : Sends a HUP signal to httpd's parent process.
# Notes    :1. Sending the HUP signal to the parent causes it to kill
#              its child processes, as would a TERM signal. However,
#              the parent doesn't exit - it re-reads its configuration
#              files, re-opens any log files, and spawns a new set of
#              child processes and continues servicing hits.
#           2. The function calls the following functions:
#               - StartHttpListener
#               - DisplayMessage
########################################################################
RestartHttpdImmediately ()
{
if [! -f ${APACHE_RUNTIME_DIR}/logs/httpd.pid]
then
    DisplayMessage E "${LISTENER_NOT_RUNNING}" N
    StartHttpListener
else
    kill -HUP `cat ${APACHE_RUNTIME_DIR}/logs/httpd.pid` 2>
${TEMP_FILE_1}
    # allow time for child processes to be killed and parent process
    # to re-start
    sleep 3
    if ps -eaf | grep "httpd" | grep -v "grep" > ${TEMP_FILE_1} 2>&1
    then
        DisplayMessage I "${LISTENER_RESTARTED}" N
        return $TRUE
    else
        ERR_MSG = `cat ${TEMP_FILE_1} | head -1`
        DisplayMessage E "${OSERROR}" Y
        return $FALSE
    fi
fi
}
########################################################################
# Name     : RestartHttpdAfterCurrentTask
# Overview : The function sends a HUP signal to the parent process of
#            httpd.
# Notes    : 1. Sending the USR1 signal to the parent causes it to
#               advise its children to exit after their current
#               request (or to exit immediately, if they're not
#               serving any requests). The parent re-reads its
#               configuration files and re-opens any log files. As
#               each child process exits, the parent replaces it with
#               a new one using the new configuration. The new process
#               begins serving new requests immediately.
#            2. The function calls following functions:
#               - StartHttpListener
#               - DisplayMessage
########################################################################
RestartHttpdAfterCurrentTask ()
{
if [! -f ${APACHE_RUNTIME_DIR}/logs/httpd.pid]
then
    DisplayMessage E "${LISTENER_NOT_RUNNING}" N
    StartHttpListener
else
    DisplayMessage I "${ADVISE_CHILD}" N
    kill -USR1 `cat ${APACHE_RUNTIME_DIR}/logs/httpd.pid` 2>
${TEMP_FILE_1}
    # allow time for the child processes to be killed and the parent
    # process to re-start
    sleep 3
    if ps -eaf | grep "httpd" | grep -v "grep" > ${TEMP_FILE_1} 2>&1
    then
        DisplayMessage I "${LISTENER_RESTARTED}" N
        return $TRUE
    else
        ERR_MSG = `cat ${TEMP_FILE_1} | head -1`
        DisplayMessage E "${OSERROR}" Y
        return $FALSE
    fi
fi
}
########################################################################
# Name     : ProcessROOTMenu
#
# Overview : The function checks the httpd configuration file for
#            the correct syntax.
#
```

```
# Notes    : 1. The function calls the following functions:
#              - StartHttpListener
#              - StopHttpListener
#              - ShowHttpdProcessDetails
#              - RestartHttpdImmediately
#              - RestartHttpdAfterCurrentTask
#              - ProcessRTOMenu
#              - ProcessEDITMenu
#              - ProcessVIEWMenu
#              - ProcessROOTMenu
#              - DisplayHotKeys
#              - ProcessExit
#              - DisplayMessage
######################################################################
ProcessROOTMenu ()
{
while true
do
    clear
    echo ""
    echo ""
    echo "\t\t###########################################################"
    echo "\t\t#                                         #"
    echo "\t\t#           ${ROOT_MENU}                  #"
    echo "\t\t#                                         #"
    echo "\t\t#    5. Start httpd                       #"
    echo "\t\t#   10. Stop httpd                        #"
    echo "\t\t#   15. Show httpd process details        #"
    echo "\t\t#   20. Restart httpd immediately         #"
    echo "\t\t#   25. Restart httpd after current tasks #"
    echo "\t\t#   30. Access runtime option menu        #"
    echo "\t\t#   35. Access edit menu                  #"
    echo "\t\t#   40. Access view menu                  #"
    echo "\t\t#                                         #"
    echo "\t\t#   99. Exit                              #"
    echo "\t\t#                                         #"
    echo "\t\t###########################################################"
    echo "\t\t Enter option (h for hotkey) ---->\c"
    read OPTION
    OPTION = `echo $OPTION | tr "a-z" "A-Z"`
    case $OPTION in
        5)    StartHttpListener;;
        10)   StopHttpListener;;
        15)   ShowHttpdProcessDetails;;
        20)   RestartHttpdImmediately;;
        25)   RestartHttpdAfterCurrentTask;;
        30)   ProcessRTOMenu;
              break;;
        35)   ProcessEDITMenu;
```

```
                break;;
         40)    ProcessVIEWMenu;
                break;;
         ROOT) ProcessROOTMenu;
                break;;
         VIEW) ProcessVIEWMenu;
                break;;
         EDIT) ProcessEDITMenu;
                break;;
         RTO)   ProcessRTOMenu;
                break;;
         H)     DisplayHotKeys;;
         99)    ProcessExit $SEC;;
         *)     DisplayMessage E "${INVALID_OPTION}" N;;
    esac
done
}
######################################################################
# Name     : CheckConfigurationFileForSyntax
#
# Overview : Checks the httpd configuration file's syntax.
#
# Notes    : 1. The function calls the following functions:
#               - FormatUnderscores
######################################################################
CheckConfigurationFileForSyntax ()
{
#
# initialize temporary file
#
HEADER = "Syntax Check"
echo "                  ${HEADER}" > ${TEMP_FILE_1}
HEADER = "Configuration File = ${HTTPD_CONFIG_FILE}"
FormatUnderscores "${HEADER}"
echo "   ${HEADER}" >> ${TEMP_FILE_1}
echo "   ${UNDERSCORE}\n\n" >> ${TEMP_FILE_1}
#
# run syntax check
#
> ${TEMP_FILE_2}
${HTTPD_DIR}/httpd -t -f ${HTTPD_CONFIG_FILE} 2> ${TEMP_FILE_2}
if grep "Syntax OK" ${TEMP_FILE_2} > /dev/null 2>&1
then
    echo "Syntax OK" >> ${TEMP_FILE_1}
else
    cat ${TEMP_FILE_2} >> ${TEMP_FILE_1}
fi
view ${TEMP_FILE_1}
}
```

```
###########################################################################
# Name     : ProcessRTOMenu
#
# Overview : The function displays and processes runtime menu
#            options.
#
# Notes    : 1. The function calls the following functions:
#               - DisplayApacheOptions
#               - CheckConfigurationFileForSyntax
#               - ProcessROOTMenu
#               - ProcessROOTMenu
#               - ProcessVIEWMenu
#               - ProcessEDITMenu
#               - ProcessRTOMenu
#               - DisplayHotKeys
#               - ProcessExit
#               - DisplayMessage
###########################################################################
ProcessRTOMenu ()
{
while true
do
    clear
    echo ""
    echo ""
    echo "\t\t################################################"
    echo "\t\t#                                              #"
    echo "\t\t#      ${RUNTIME_MENU}                         #"
    echo "\t\t#                                              #"
    echo "\t\t#   5. Show run-time options                   #"
    echo "\t\t#  10. Show version number                     #"
    echo "\t\t#  15. Show compile settings                   #"
    echo "\t\t#  20. Show parsed settings                    #"
    echo "\t\t#  25. Show modules compiled in                #"
    echo "\t\t#  30. Show configuration directives available #"
    echo "\t\t#  35. Run syntax check for configuration file #"
    echo "\t\t#                                              #"
    echo "\t\t#  80. Main menu                               #"
    echo "\t\t#  99. Exit                                    #"
    echo "\t\t#                                              #"
    echo "\t\t################################################"
    echo "\t\t Enter option (h for hotkey) ---->\c"
    read OPTION
    OPTION = `echo $OPTION | tr "a-z" "A-Z"`
    case $OPTION in
        5)    DisplayApacheOptions "h";;
        10)   DisplayApacheOptions "v";;
        15)   DisplayApacheOptions "V";;
        20)   DisplayApacheOptions "S";;
        25)   DisplayApacheOptions "l";;
        3Ø)   DisplayApacheOptions "L";;
        35)   CheckConfigurationFileForSyntax;;
        8Ø)   ProcessROOTMenu;
              break;;
        ROOT) ProcessROOTMenu;
              break;;
        VIEW) ProcessVIEWMenu;
              break;;
        EDIT) ProcessEDITMenu;
              reak;;
        RTO)  ProcessRTOMenu;
              break;;
        H)    DisplayHotKeys;;
        99)   ProcessExit $SEC;;
        *)    DisplayMessage E "${INVALID_OPTION}" N;;
    esac
done
}
######################################################################
# Name     : GetFileName
#
# Overview : The function gets the name of the file that's the
#            argument for this function.
#
# Input    : File type:
#            - ERROR_LOG
#            - ACCESS_LOG
#
# Returns  : $TRUE
#            $FALSE
######################################################################
GetFileName ()
{
#
# assign parameter
#
FILE_TYPE = "$1"
case ${FILE_TYPE} in
    ERROR_LOG)
     #
        # get error log location from config file
        #
        if grep -i "ErrorLog" ${HTTPD_CONFIG_FILE} > ${TEMP_FILE_1} 2>&1
        then
        #
        # may have found ErrorLog directive
        #
        cat ${TEMP_FILE_1} | while read LINE
        do
            #
            # examine line for comment
            #
            FIRST_CHAR = `echo "${LINE}" | cut -c1-1`;
            if ["${FIRST_CHAR}" = "#"]
            then
                continue;
            fi;
            #
            # scan line for directive
            #
            ERROR_LOG_DIRECTIVE = `echo "${LINE}" | cut -d' ' -f1`;
ERROR_LOG_DIRECTIVE = `echo "${ERROR_LOG_DIRECTIVE}" | tr "a-z" "A-z"`;
            if ["${ERROR_LOG_DIRECTIVE}" ! = "ERRORLOG"]
            then
                continue;
            fi;
            #
            # found ErrorLog directive
            #
            HTTPD_ERROR_LOG_FILE = `echo "${LINE}" | cut -c9-`;
 HTTPD_ERROR_LOG_FILE = `echo "${HTTPD_ERROR_LOG_FILE}" | sed s/'"//g`;
            return $TRUE;
        done;
        else
        #
        # file must be in default location
        #
        HTTPD_ERROR_LOG_FILE = ${DEFAULT_HTTPD_ERROR_LOG_FILE};
        return $TRUE;
        fi;
        return $TRUE;;
    ACCESS_LOG)
        #
        # get error log location from config file
        #
       if grep -i "CustomLog" ${HTTPD_CONFIG_FILE} > ${TEMP_FILE_1} 2>&1
        then
            #
            # may have found CustomLog directive
            #
            cat ${TEMP_FILE_1} | while read LINE
            do
                #
                # examine line for comment
                #
                FIRST_CHAR = `echo "${LINE}" | cut -c1-1`;
                if ["${FIRST_CHAR}" = "#"]
                then
                    continue;
                fi;
                #
                # scan line for directive
                #
                ACCESS_LOG_DIRECTIVE = `echo "${LINE}" | cut -d' ' -f1`;
                ACCESS_LOG_DIRECTIVE = `echo "${ACCESS_LOG_DIRECTIVE}" |
tr "a-z" "A-z"`;
                if ["${ACCESS_LOG_DIRECTIVE}" ! = "CUSTOMLOG"]
                then
                    continue;
                fi;
                #
                # found CustomLog directive
                #
                HTTPD_ACCESS_LOG_FILE = `echo "${LINE}" | cut -c10-`;
                HTTPD_ACCESS_LOG_FILE = `echo "${HTTPD_ACCESS_LOG_FILE}"
| sed s/\"//g | awk {'print $1'}`;
             FIRST_CHAR = `echo "${HTTPD_ACCESS_LOG_FILE}" | cut -c1-1`;
                if ["${FIRST_CHAR}" ! = "/"]
                then
                    HTTPD_ACCESS_LOG_FILE = "${APACHE_RUNTIME_DIR}/
${HTTPD_ACCESS_LOG_FILE}";
                fi
                return $TRUE;
            done;
        else
            return $FALSE;
        fi;;
esac
}
#######################################################################
# Name     : ProcessMainDirectory
#
# Overview : The function lists all the files and subdirectories of a
#            given top-level directory.
#
# Input    : Top-level directory name.
#
# Notes    : 1. The function calls the following functions:
#               - ProcessSubDirectory
#               - FormatUnderscores
#######################################################################
ProcessMainDirectory ()
{
DisplayMessage I "${WORKING}" N
INDEX = 0
LEVEL1_SUB_DIR_FILE = /tmp/dl_$$_$INDEX.dir
```

```
#
# assign parameter
#
P_DIR = "$1"
HEADER = "File Listing for $P_DIR"
FormatUnderscores "${HEADER}"
echo "\n${HEADER}" >> ${TEMP_FILE_1}
echo "${UNDERSCORE}" >> ${TEMP_FILE_1}
#
# switch to the directory
#
cd $P_DIR
ls -1 > ${TEMP_FILE_3}
cat ${TEMP_FILE_3} | while read FILE_NAME
do
    if [-d ${FILE_NAME}]
    then
        #
        # file is a directory
        #
        echo "${P_DIR}/$FILE_NAME" >> $LEVEL1_SUB_DIR_FILE
        echo "Directory -----> ${P_DIR}/$FILE_NAME" >> ${TEMP_FILE_1}
    else
        echo "File ----------> ${P_DIR}/$FILE_NAME" >> ${TEMP_FILE_1}
    fi
done
ProcessSubDirectory "${LEVEL1_SUB_DIR_FILE}"
return $TRUE
}
#######################################################################
# Name     : ProcessSubDirectory
#
# Overview : The function processes the files and subdirectories of a
#            subdirectory recursively.
#
# Notes    : 1. The variable SUB_DIR_FILE holds the name of a file
#               that contains the names of directories that need to
#               be processed.
#
#            2. The variable DIR_TO_BE_PROCESSED holds the name of a
#               file that contains a list of the subdirectories in
#               the directory being processed.
#
#            3. The function calls the following function:
#               - FormatUnderscores
#######################################################################
ProcessSubDirectory ()
{
#
```

```
# assign parameter
#
SUB_DIR_FILE = "$1"
INDEX = `expr $INDEX + 1`
DIR_TO_BE_PROCESSED = "/tmp/dl_$$_$INDEX.dir"
cat ${SUB_DIR_FILE} | while read DIR
do
    HEADER = "File listing for $DIR"
    FormatUnderscores "${HEADER}"
    echo "\n${HEADER}" >> ${TEMP_FILE_1}
    echo "${UNDERSCORE}" >> ${TEMP_FILE_1}
    #
    # switch to the directory
    #
    cd $DIR
    ls -1 > ${TEMP_FILE_3}
    cat ${TEMP_FILE_3} | while read FILE_NAME
    do
        if [-d ${FILE_NAME}]
        then
            #
            # file is a directory
            #
            echo "$DIR/$FILE_NAME" >> ${DIR_TO_BE_PROCESSED}
            echo "Directory -----> ${DIR}/$FILE_NAME" >> ${TEMP_FILE_1}
        else
            echo "File ----------> ${DIR}/$FILE_NAME" >> ${TEMP_FILE_1}
        fi
        done
        if [-s ${DIR_TO_BE_PROCESSED}]
        then
            ProcessSubDirectory ${DIR_TO_BE_PROCESSED}
        fi
done
#
# delete temporary files
#
rm -f /tmp/dl_$$_*.dir
return $TRUE
}
#######################################################################
# Name     : DisplayListOfValues
#
# Overview : Displays a list of file types provided.
#
# Input    : File types
#            - APACHE_SOURCE
#            - HTML_SOURCE
#
```

```
# Notes    : 1. The function calls the following functions:
#              - ProcessMainDirectory
#              - FormatUnderscores
####################################################################
DisplayListOfValues ()
{
#
# assign parameter
#
FILE_TYPE = "$1"
case ${FILE_TYPE} in
    APACHE_SOURCE)
        #
        # initialize temporary file
        #
        HEADER = "List of values for Apache modules";
        FormatUnderscores "${HEADER}";
        echo "  ${HEADER}     " > ${LOV_FILE_1};
        echo "  ${UNDERSCORE}\n" >> ${LOV_FILE_1};
        echo "To select a source, delete a line containing a file name"
>> ${LOV_FILE_1};
        echo "and save the file\n" >> ${LOV_FILE_1};
        #
        # does the list of values file exist for this session?
        #
        if [! -s ${APACHE_SOURCE_LIST}]
        then
            #
            # include sources from all source directories
            # in the file $APACHE_SOURCE_LIST
            #
            ProcessMainDirectory "${APACHE_SOURCE_DIR}"
            cp ${TEMP_FILE_1} ${APACHE_SOURCE_LIST}
            cat ${APACHE_SOURCE_LIST} >> ${LOV_FILE_1};
            cp ${LOV_FILE_1} ${LOV_FILE_2};
        else
            cat ${APACHE_SOURCE_LIST} >> ${LOV_FILE_1};
            cp ${LOV_FILE_1} ${LOV_FILE_2};
        fi;
        view ${LOV_FILE_2};
        FILE_TO_VIEW = `diff ${LOV_FILE_1} ${LOV_FILE_2} | tail -1 | awk
{'print $4'}`;
        return $TRUE;;
    HTML_SOURCE)
        #
        # initialize temporary file
        #
        HEADER = "List of Values for HTML Documents";
        FormatUnderscores "${HEADER}";
        echo "  ${HEADER}     " > ${LOV_FILE_1};
        echo "  ${UNDERSCORE}\n" >> ${LOV_FILE_1};
        echo "To Select a Source, Delete a Line and Save the File\n">>
${LOV_FILE_1};
        if [! -s ${HTML_SOURCE_LIST}]
        then
            #
            # include sources from all source directories
            # in the file $HTML_SOURCE_LIST
            #
            ProcessMainDirectory "${HTML_SOURCE_DIR}"
            cp ${TEMP_FILE_1} ${HTML_SOURCE_LIST}
            cat ${HTML_SOURCE_LIST} >> ${LOV_FILE_1};
            cp ${LOV_FILE_1} ${LOV_FILE_2};
        else
            cat ${HTML_SOURCE_LIST} >> ${LOV_FILE_1};
            cp ${LOV_FILE_1} ${LOV_FILE_2};
        fi;
        view ${LOV_FILE_2};
        FILE_TO_VIEW = `diff ${LOV_FILE_1} ${LOV_FILE_2} | tail -1 | awk
{'print $4'}`;
        return $TRUE;;
esac
}
######################################################################
# Name     : ViewFile
#
# Overview : The function allows the user to view various log files.
#
# Input    : File Type
#            - HTTPD_CONFIG
#            - HTTPD_CONFIG_TEMPLATE
#            - HTTPD_MAKEFILE_TEMPLATE
#            - HTTPD_MAKE_CONFIG_FILE
#            - APACHE_SOURCE
#            - HTML_SOURCE
#            - STATIC_ACCESS_LOG
#            - DYNAMIC_ACCESS_LOG
#            - ERROR_LOG
#
# Notes    : 1. The function allows the user to view the access
#               log file dynamically. To quit viewing, use
#               ctrl-c. In order for HandleInterrupt () to ignore
#               this interrupt, it sets the variable $FUNCTION_NAME
#               which is read by the interrupt handling function.
#
#            2. The function calls the following functions:
#               - FormatUnderscores
#               - DisplayListOfValues
#                - DisplayMessage
#                - GetFileName
######################################################################
ViewFile ()
{
#
# set the function name variable
#
FUNCTION_NAME = "ViewFile"
#
# assign parameter
#
VIEW_FILE_TYPE = "$1"
case ${VIEW_FILE_TYPE} in
    HTTPD_CONFIG)
        FILE_TO_VIEW = "${HTTPD_CONFIG_FILE}";
        #
        # initialize temporary file
        #
        HEADER = "Httpd Configuration file, ${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "  ${HEADER}     " > ${TEMP_FILE_1};
        echo "  ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    HTTPD_CONFIG_TEMPLATE)
        FILE_TO_VIEW = "${HTTPD_CONFIG_TEMPLATE}";
        #
        # initialize temporary file
        #
        HEADER = "Httpd Configuration Template File"
        echo "      ${HEADER}" > ${TEMP_FILE_1};
        HEADER = "${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "      ${HEADER}" >> ${TEMP_FILE_1};
        echo "      ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    HTTPD_MAKEFILE_TEMPLATE)
        FILE_TO_VIEW = "${HTTPD_MAKEFILE_TEMPLATE}";
        #
        # initialize temporary file
        #
        HEADER = "Httpd Makefile Template File"
        echo "      ${HEADER}" > ${TEMP_FILE_1};
        HEADER = "${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "      ${HEADER}" >> ${TEMP_FILE_1};
        echo "      ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    HTTPD_MAKE_CONFIG_FILE)
        FILE_TO_VIEW = "${HTTPD_MAKE_CONFIG_FILE}";
        #
        # initialize temporary file
        #
        HEADER = "Httpd Makefile Configuration File"
        echo "      ${HEADER}" > ${TEMP_FILE_1};
        HEADER = "${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "      ${HEADER}" >> ${TEMP_FILE_1};
        echo "      ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    APACHE_SOURCE)
        DisplayListOfValues "APACHE_SOURCE";
        if ["${FILE_TO_VIEW}" = ""]
        then
            DisplayMessage E "${NO_FILE_SELECTED}" N;
            return $FALSE;
        fi;
        if file ${FILE_TO_VIEW} | grep "text" > /dev/null 2>&1
        then
            :;
        else
            DisplayMessage E "${NOT_TEXT_FILE}" N;
            return $FALSE;
        fi;
        #
        # initialize temporary file
        #
        HEADER = "Source for Module ${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "   ${HEADER}" > ${TEMP_FILE_1};
        echo "   ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    HTML_SOURCE)
        DisplayListOfValues "HTML_SOURCE";
        if ["${FILE_TO_VIEW}" = ""]
        then
            DisplayMessage E "${NO_FILE_SELECTED}" N;
            return $FALSE;
        fi;
        if file ${FILE_TO_VIEW} | grep "text" > /dev/null 2>&1
        then
            :;
        else
            DisplayMessage E "${NOT_TEXT_FILE}" N;
            return $FALSE;
        fi;
        #
        # initialize temporary file
        #
        HEADER = "Source for Module ${FILE_TO_VIEW}";
        FormatUnderscores "${HEADER}";
        echo "   ${HEADER}" > ${TEMP_FILE_1};
        echo "   ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    ERROR_LOG)
```

```
            if ! GetFileName "ERROR_LOG"
            then
            return $FALSE;
            fi;
            #
            # initialize temporary file
            #
            FILE_TO_VIEW = "${HTTPD_ERROR_LOG_FILE}";
            HEADER = "Error Log Details From ${FILE_TO_VIEW}";
            FormatUnderscores "${HEADER}";
            echo "   ${HEADER}" > ${TEMP_FILE_1};
            echo "   ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    STATIC_ACCESS_LOG)
            if ! GetFileName "ACCESS_LOG"
            then
            return $FALSE;
            fi;
            #
            # initialize temporary file
            #
            FILE_TO_VIEW = "${HTTPD_ACCESS_LOG_FILE}";
            HEADER = "Access Log Details From ${FILE_TO_VIEW}";
            FormatUnderscores "${HEADER}";
            echo "   ${HEADER}" > ${TEMP_FILE_1};
            echo "   ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
    DYNAMIC_ACCESS_LOG)
            if ! GetFileName "ACCESS_LOG"
            then
            return $FALSE;
            fi;
            #
            # initialize temporary file
            #
            FILE_TO_VIEW = "${HTTPD_ACCESS_LOG_FILE}";
            HEADER = "Access Log Details From ${FILE_TO_VIEW}";
            FormatUnderscores "${HEADER}";
            echo "   ${HEADER}" > ${TEMP_FILE_1};
            echo "   ${UNDERSCORE}\n" >> ${TEMP_FILE_1};;
esac
#
# does the file exist?
#
if [! -f ${FILE_TO_VIEW}]
then
    DisplayMessage E "${FILE_NOT_FOUND}" N
    return $FALSE
fi
#
# establish file type
#
```

```
if ["${VIEW_FILE_TYPE}" = "DYNAMIC_ACCESS_LOG"]
then
    DisplayMessage I "${QUIT_ACCESS_LOG_VIEWING}" N
    clear
    FUNCTION_NAME = "ViewFile"
    tail -f ${FILE_TO_VIEW}
else
    #
    # make a copy of the file before viewing
    #
    cat ${FILE_TO_VIEW} >> ${TEMP_FILE_1}
    view ${TEMP_FILE_1}
    #
    # offer option to print
    #
    PrintFile "${TEMP_FILE_1}"
fi
}
######################################################################
# Name     : ProcessVIEWMenu
#
# Overview : The function displays and processes VIEW menu options.
#
# Notes    : 1. The function calls the following functions:
#               - ViewFile
#               - ProcessROOTMenu
#               - ProcessVIEWMenu
#               - ProcessEDITMenu
#               - ProcessRTOMenu
#               - DisplayHotKeys
#               - DisplayMessage
######################################################################
ProcessVIEWMenu ()
{
while true
do
    clear
    echo ""
    echo ""
    echo "\t\t##############################################"
    echo "\t\t#                                            #"
    echo "\t\t#       ${VIEW_MENU}                         #"
    echo "\t\t#                                            #"
    echo "\t\t#     5. View httpd error log                #"
    echo "\t\t#    10. View httpd access log statically    #"
    echo "\t\t#    15. View httpd access log dynamically   #"
    echo "\t\t#    20. View httpd configuration file       #"
    echo "\t\t#    25. View configuratin template          #"
    echo "\t\t#    30. View makefile template              #"
    echo "\t\t#    35. View makefile configuration file    #"
    echo "\t\t#    40. View Apache module sources          #"
    echo "\t\t#    45. View html document sources          #"
    echo "\t\t#                                            #"
    echo "\t\t#    80. Main menu                           #"
    echo "\t\t#    99. Exit                                #"
    echo "\t\t#                                            #"
    echo "\t\t##############################################"
    echo "\t\t Enter option (h for hotkey) ---->\c"
    read OPTION
    OPTION = `echo $OPTION | tr "a-z" "A-Z"`
    case $OPTION in
        5)    ViewFile "ERROR_LOG";;
        10)   ViewFile "STATIC_ACCESS_LOG";;
        15)   ViewFile "DYNAMIC_ACCESS_LOG";;
        20)   ViewFile "HTTPD_CONFIG";;
        25)   ViewFile "HTTPD_CONFIG_TEMPLATE";;
        30)   ViewFile "HTTPD_MAKEFILE_TEMPLATE";;
        35)   ViewFile "HTTPD_MAKE_CONFIG_FILE";;
        40)   ViewFile "APACHE_SOURCE";;
        45)   ViewFile "HTML_SOURCE";;
        80)   ProcessROOTMenu;
              break;;
        ROOT) ProcessROOTMenu;
              break;;
        VIEW) ProcessVIEWMenu;
              break;;
        EDIT) ProcessEDITMenu;
              break;;
        RTO)  ProcessRTOMenu;
              break;;
        H)    DisplayHotKeys;;
        99)   ProcessExit $SEC;;
        H)    DisplayHotKeys;;
        *)    DisplayMessage E "${INVALID_OPTION}" N;;
    esac
done
}
#######################################################################
# Name     : EditHttpdMakeConfigurationFile
#
# Overview : Allows the user to edit the httpd configuration file.
#
# Notes    : 1. The function calls the following functions:
#               - DisplayMessage
#               - PrintFile
#######################################################################
EditHttpdMakeConfigurationFile ()
{
#
# does the file exist?
```

```
#
if [! -f ${HTTPD_MAKE_CONFIG_FILE}]
then
    FILE = "${HTTPD_MAKE_CONFIG_FILE}"
    DisplayMessage E "${CONF_FILE_NOT_FOUND}" Y
    return $FALSE
fi
#
# edit the file
#
vi ${HTTPD_MAKE_CONFIG_FILE}
#
# offer option to print
#
PrintFile "${HTTPD_MAKE_CONFIG_FILE}"
}
#######################################################################
# Name     : EditHttpdConfigurationFile
#
# Overview : Allows the user to edit the httpd configuration file.
#
# Notes    : 1. The function calls the following functions:
#               - DisplayMessage
#               - PrintFile
#######################################################################
EditHttpdConfigurationFile ()
{
#
# does the file exist?
#
if [! -f ${HTTPD_CONFIG_FILE}]
then
    FILE = "${HTTPD_CONFIG_FILE}"
    DisplayMessage E "${CONF_FILE_NOT_FOUND}" Y
    return $FALSE
fi
#
# edit the file
#
vi ${HTTPD_CONFIG_FILE}
#
# offer option to print
#
PrintFile "${HTTPD_CONFIG_FILE}"
}
#######################################################################
# Name     : EditHtmlSource
#
# Overview : Allows the user to edit an html source file.
```

```
#
# Notes    : 1. The function calls the following functions:
#               - DisplayListOfValues
#               - PrintFile
######################################################################
EditHtmlSource ()
{
DisplayListOfValues "HTML_SOURCE"
if ["${FILE_TO_VIEW}" = ""]
then
    DisplayMessage E "${NO_FILE_SELECTED}" N
    return $FALSE
fi
if file ${FILE_TO_VIEW} | grep "text" > /dev/null 2>&1
then
    :
else
    DisplayMessage E "${NOT_TEXT_FILE}" N
    return $FALSE
fi
#
# allow user to edit the file
#
vi ${FILE_TO_VIEW}
PrintFile ${FILE_TO_VIEW}
}
######################################################################
# Name     : EditApacheSource
#
# Overview : Allows the user to edit an Apache source file.
#
# Notes    : 1. The function calls the following functions:
#               - PrintFile
#               - DisplayListOfValues
######################################################################
EditApacheSource ()
{
DisplayListOfValues "APACHE_SOURCE"
if ["${FILE_TO_VIEW}" = ""]
then
    DisplayMessage E "${NO_FILE_SELECTED}" N
    return $FALSE
fi
if file ${FILE_TO_VIEW} | grep "text" > /dev/null 2>&1
then
    :
else
    DisplayMessage E "${NOT_TEXT_FILE}" N
    return $FALSE
fi
# allow user to edit the file
vi ${FILE_TO_VIEW}
PrintFile ${FILE_TO_VIEW}
}
######################################################################
# Name     : ProcessEDITMenu
# Overview : The function displays and processes EDIT menu options.
# Notes    :1. The function calls the following functions:
#               - DisplayMessage I
#               - EditHttpdConfigurationFile
#               - EditHtmlSource
#               - EditApacheSource
#               - ProcessROOTMenu
#               - ProcessVIEWMenu
#               - ProcessEDITMenu
#               - ProcessRTOMenu
#               - DisplayHotKeys
#               - ProcessExit
#               - DisplayMessage
######################################################################
ProcessEDITMenu ()
{
while true
do
    clear
    echo ""
    echo ""
    echo "\t\t############################################"
    echo "\t\t#                                          #"
    echo "\t\t#         ${EDIT_MENU}                     #"
    echo "\t\t#                                          #"
    echo "\t\t#     5. Edit make configuration file      #"
    echo "\t\t#    10. Edit httpd configuration file     #"
    echo "\t\t#    15. Edit html sources                 #"
    echo "\t\t#    20. Edit Apache sources               #"
    echo "\t\t#                                          #"
    echo "\t\t#    80. Main menu                         #"
    echo "\t\t#    99. Exit                              #"
    echo "\t\t############################################"
    echo "\t\t Enter option (h for hotkey) ---->\c"
    read OPTION
    OPTION = `echo $OPTION | tr "a-z" "A-Z"`
    case $OPTION in
        5)    EditHttpdMakeConfigurationFile;;
        10)   EditHttpdConfigurationFile;;
        15)   EditHtmlSource;;
        20)   EditApacheSource;;
        80)   ProcessROOTMenu;
              break;;
        ROOT) ProcessROOTMenu;
              break;;
        VIEW) ProcessVIEWMenu;
              break;;
        EDIT) ProcessEDITMenu;
              break;;
        RTO)  ProcessRTOMenu;
              break;;
        H)    DisplayHotKeys;;
        99)   ProcessExit $SEC;;
        *)    DisplayMessage E "${INVALID_OPTION}" N;;
    esac
done
}
######################################################################
# Name     : PerformSanityCheck
# Overview : Checks the validity of all directories defined.
# Returns  : TRUE if all the checks are OK
#            FALSE otherwise
######################################################################
PerformSanityCheck ()
{
# check Apache source directory
if [! -d "${APACHE_ROOT_DIR}"]
then
    DIR_NAME = "${APACHE_ROOT_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}" Y
    return $FALSE
fi
# check Apache run-time root directory
if [! -d "${APACHE_RUNTIME_DIR}"]
then
    DIR_NAME = "${APACHE_RUNTIME_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}" Y
    return $FALSE
fi
# check Apache run-time conf directory
if [! -d "${APACHE_RUNTIME_DIR}/conf"]
then
    DIR_NAME = "${APACHE_RUNTIME_DIR}/conf" Y
    DisplayMessage E "${DIR_NOT_EXIST}"
    return $FALSE
fi
# check Apache run-time logs directory
if [! -d "${APACHE_RUNTIME_DIR}/logs"]
then
    DIR_NAME = "${APACHE_RUNTIME_DIR}/logs" Y
    DisplayMessage E "${DIR_NOT_EXIST}" N
    return $FALSE
fi
```

```
# check Apache run-time docs directory
if [! -d  "${APACHE_RUNTIME_DIR}/htdocs"]
then
    DIR_NAME = "${APACHE_RUNTIME_DIR}/htdocs"
    DisplayMessage E "${DIR_NOT_EXIST}" N
    return $FALSE
fi
# check httpd directory
if [! -d "${HTTPD_DIR}"]
then
    DIR_NAME = "${HTTPD_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}" N
    return $FALSE
fi
return $TRUE
}
########################################################################
# Name     : main
# Overview : The function implements the processing structure.
# Notes    : 1. The function calls the following functions:
#               - InitializeVariables
#               - RootUser
#               - DisplayMessage
#               - ProcessExit
#               - PerformSanityCheck
#               - ProcessROOTMenu
########################################################################
main ()
{
InitializeVariables
trap "HandleInterrupt " $SIGINT $SIGTERM $SIGHUP $SIGTSTP
if ! RootUser
then
    DisplayMessage E "${ROOT_USER}" N
    ProcessExit $FEC
fi
if ! PerformSanityCheck
then
    ProcessExit $FEC
fi
ProcessROOTMenu
}
# invoke main
main
```

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

Understanding the *tr* command

Suppose you had text files, such as documentation or source files, that required massive, global substitutions at the character level.

Consider the following three examples:

1 A source code file in which you need to change every occurrence of the tab character to a newline character.

2 A help text document in which you want to delete all occurrences of a certain bullet character.

3 A README file in which you need to reduce all multiple blank lines to a single occurrence each.

There are a few ways you can accomplish this in AIX. The most tedious would probably be to edit the files with the ed or vi editors and to make all the changes manually. Though some editors allow certain global changes, the manual process can be error prone.

A more efficient way would be to use the *tr* command. The *tr*, or translate, command can perform one or more operations such as these in one simple step from the command line without having to edit the input file. In certain cases it may be desirable to preserve the integrity of the input file by not editing it.

There are three basic operations that the *tr* command can perform:

- One-for-one translation of characters – characters found in the input file that match those in one string are replaced with the corresponding character found in the same position in another string, and are written to the output file.
- Character deletion – characters found in the input file that match those in a string are deleted and the remaining characters are written to the output file.
- Sequence removal – multiple occurrences of characters found in the input file that match those in the string specification are

reduced, or ‘squeezed’, into one occurrence each before being written to the output file.

Multiple operations can be combined into one at the command line if the proper flags are specified. It is common to use the redirection symbols (less than and greater than signs) to direct data from an input file into the *tr* command and out to an output file. The typical syntax of the *tr* command is shown below:

```
tr  flags  String1  String2  <  infile  >  outfile
```

where:

- flags – an optional flag or flags used to enhance the *tr* operation.
- String1 – usually the desired source characters to be changed.
- String2 – usually the target characters changed to by *tr*.
- infile – the file containing the input data.
- outfile - the file to which the *tr* command is to write.

FLAGS FOR THE *TR* COMMAND

The following section describes how to use flags to specify how the *tr* command is to operate:

- -c – considers all characters with the exception of those specified in String1. Using the *c* flag tells the *tr* command to use the complement, or ‘all but’ those in the character set represented by String1, as the characters to process.
- -d – deletes characters from the input file found in String1 before writing to the output file.
- -s – reduces multiple occurrences of String1 or String2 characters into only one occurrence.

The flags used with the *tr* command are summarized below:

```
+------------+-----------------------------------------------+
|  FLAG      |      DESCRIPTION                              |
+------------+-----------------------------------------------+
|  -c        | Consider 'all but' those characters           |
|            | specified by String1.                         |
+-------------+---------------------------------------------+
|   -d        |  Delete all characters matching those found   |
|             |  in String1.                                  |
+-------------+---------------------------------------------+
|   -s        |  Squeeze multiple occurrences of characters   |
|             |  found in String1 and String2 into one.       |
+-------------+---------------------------------------------+
```

SOME QUICK EXERCISES

Here are some quick exercises to see how the *tr* command works. Performing these steps on your AIX workstation before continuing reading can help you to understand the remainder of the article better. Each step has you piping the string 'Hello World' into the *tr* command, using some of the string and flag specifications explained above:

1 Entering:

```
echo "Hello World" | tr o 0
```

yields:

```
Hell0 W0rld
```

all 'o's have been changed to zeros.

2 Entering:

```
echo "Hello World" | tr -d l
```

yields:

```
Heo Word
```

all 'l's have been deleted.

3 Entering:

```
echo "Hello World" | tr -s l
```

yields:

```
Helo World
```

the double 'l's have been reduced to one; the single l is unaffected.

STRING SPECIFICATION

There are different methods of specifying String1 and String2 that can help you to get the desired results. Some specifications require the use of square brackets []. To leave them off would give unpredictable results.

- char1-char2 tells *tr* to consider the characters within the specified range.

  For example, a-z is lowercase alphabetic characters from a to z. 0-9 would be numeric digits from 0 through 9. L-P would be the uppercase letters L, M, N, O, and P.

- [char*] makes String2 the same length as String1 by repeating the last character specified in String2.

  For example, if String1 were [abcdefg] and String2 was specified as [x*], the asterisk would tell the *tr* command to make String2 'xxxxxxx' to make it the same length as String1. This ensures that there is a one-for-one character representation for translation between String1 and String2 without having to specify all the characters in String2. In this case, any character found in String1 in the input file would be translated into an x.

- [char*num] repeats the specified character by the specified number.

  Examples:

  > [x*3] would be 'xxx'
  >
  > [Z*5] would be 'ZZZZZ'
  >
  > [#*4] would be '####'
  >
  > [' '*5] would be '   '.

  If the first digit of num is a zero, then num is considered an octal, otherwise num is considered decimal. For example, [x*11] would be 'xxxxxxxxxxx' (eleven 'x's) but [x*011] would be 'xxxxxxxxx' (nine 'x's) because an octal 011 equals a decimal 9. Be careful with programs that insert variables into your string values. A program that pads decimal values with leading zeroes will cause *tr* to treat the value as octal.

- [:class:] uses all the characters defined in the specified class.

  The following are classes you can use with the *tr* command:

  [:alnum:] alpha-numeric (letters and numbers)

  [:alpha:] letters only (lowercase a-z and uppercase A-Z)

  [:blank:] spaces and tabs

  [:cntrl:] control characters

  [:digit:] numbers 0-9

  [:graph:] printable, visible characters

  [:lower:] lowercase letters (a-z)

  [:print:] printable, non-control characters

  [:punct:] punctuation characters (excludes alphanumeric, control, or space characters)

  [:space:] blank, tab, newline, vertical tab, formfeed, return

  [:upper:] uppercase letters (A-Z)

  [:xdigit:] hexadecimal characters (0-9 a-f A-F).

- \oct indicates an octal representation.

  For example, '\12' or '\012' will tell *tr* that you are specifying the new-line character. Using the backslash character to specify an octal integer precludes the need to precede the digits with zeroes.

- \cntlchar tells the *tr* command to consider the value a control character.

  The following values can be used:

cntl char	*octal equiv*	*description*
'\a'	'\007'	Alert (bell)
'\b'	'\010'	Backspace
'\f'	'\014'	Form-feed
'\n'	'\012'	New line
'\r'	'\015'	Carriage return

‘\t’	‘\011’	Tab
‘\v’	‘\013’	Vertical tab

- \char specifies a character whose normal value you desire to ‘escape’. Use the backslash character to escape certain characters that have a special meaning to the *tr* command.

  Examples:

  ‘\[’ escape the left bracket without considering it as the beginning of a string sequence.

  ‘\-’ escape the minus without considering it as a range separator.

  ‘\\’ escape the backslash without considering it as the escape character.

EXAMPLES

With the three distinct types of operation the *tr* command can handle, there are several variations of flags and string combinations to consider. Some invocations use only String1 and others use both String1 and String2. The following section gives examples of each possible variation using the flags described in this article.

Note: the infile and outfile examples specify control characters such as \t and \n to represent the actual tab and newline characters that would be in the file.

The various methods that the tr command can be used are summarized below:

```
+-------------+-----------------------+-----------------------------+
| Example 1   | tr -d   String1       | delete String1 from input file|
+-------------+-----------------------+-----------------------------+
| Example 2   | tr -s   String1       |squeeze String1 from input file|
+-------------+-----------------------+-----------------------------+
| Example 3   | tr -cd  String1       | delete all but String1 from  |
|             |                       |  input file                  |
+-------------+-----------------------+-----------------------------+
| Example 4   | tr -cs  String1       | squeeze all but String1 from |
|             |                       |  input file                  |
+-------------+-----------------------+-----------------------------+
| Example 5   | tr     String1 String2 | translate String1 to String2 |
+-------------+-----------------------+-----------------------------+
```

Example 6	tr -c String1 String2	translate all but String1 to String2
Example 7	tr -s String1 String2	squeeze String1 from input, translate, and squeeze String2 from output
Example 8	tr -cs String1 String2	squeeze all but String1 from input, translate, and squeeze String2 from output
Example 9	tr -ds String1 String2	delete String1 from input, and squeeze String2 from output
Example 10	tr -cds String1 String2	delete all but String1 from input, and squeeze String2 from output

EXAMPLE 1

Syntax: tr -d String1.

Purpose: delete String1 from input file.

Example: tr -d '\007' < infile > outfile.

Result: deletes each occurrence of the bell character from the input file and writes results to the output file. This would prevent bell alerts if the output file were displayed to the screen using the *cat* command.

EXAMPLE 2

Syntax: tr -s String1.

Purpose: squeeze String1 from input file.

Example: tr -s '\f' < infile > outfile.

Result: squeezes multiple sequences of the ‘formfeed’ control character into one. This would limit the numbers of multiple blank pages between text pages to one in the output file.

EXAMPLE 3

Syntax: tr -cd String1.

Purpose: delete all but String1 from input file.

Example: tr -cd [:xdigit:] < infile > outfile.

Result: deletes everything from the input file except for hexadecimal characters and writes results to the output file.

infile: ====> 00F2FAD0, 00F2ACD0, 00F2BBA0 <====

outfile: 00F2FAD000F2ACD000F2BBA0

In this example, the *tr* command deleted all of the comma, space, equals, and redirection symbols from the input file, leaving raw hex characters.

Use caution when specifying the -*c* flag. Consider if you executed the previous command against the following:

infile: These 4 Bytes are HEX: 00F2FAD0

outfile: ee4BeaeE00F2FAD0

In this case, the alphanumeric characters in the text that are also hexadecimal characters will be processed and written to the output file as hex characters.

EXAMPLE 4

Syntax: tr -cs String1.

Purpose: squeeze all but String1 from input file.

Example: tr -cs [:alnum:] < infile > outfile.

Result: squeezes any repeating sequences of characters other than alphanumeric characters in the input file into a single occurrence and writes results to the output file.

infile: GOOD DATA\t\t\t00223\n\nBETTER DATA\t\t\t000445\n\nBEST DATA\t\t\t000667

outfile: GOOD DATA\t000223\nBETTER DATA\t000445\nBEST DATA\t000667

In this example, the *tr* command reduced the sequences of tabs and newlines to one occurrence each. The alphanumerics, including any sequences, were left unchanged.

EXAMPLE 5

Syntax: tr String1 String2.

Purpose: translate String1 to String2.

Example: tr 'S' '$' < infile > outfile.

Result: translates every occurrence of the uppercase ‘S’ in the input file to a dollar sign and writes results to the output file.

infile: Seven SESSIONS serving Seven Seminars

outfile: $even $E$$ION$ serving $even $eminars

In this example, the *tr* command changed all uppercase ‘S’s to a dollar sign, ignoring all lowercase ‘S’s.

EXAMPLE 6

Syntax: tr -c String1 String2.

Purpose: translate all but String1 to String2.

Example: tr -c [:alpha:] [-*] < infile > outfile.

Result: translates every character found in the input file to a ‘-’ (minus sign) except upper and lower case letters, and writes results to the output file.

infile: 100 Cents = 1 Dollar.

outfile: ----Cents-----Dollar--

In this example, the *tr* command changed all numbers, spaces, punctuation, and the carriage return to the ‘-’ (minus sign).

EXAMPLE 7

Syntax: tr -s String1 String2.

Purpose: squeeze String1 from input, translate, and squeeze String2 from output.

Example: tr -s '\t' ' ' < infile > outfile.

Result: squeezes sequences of the tab character from the input file into one occurrence each, translates each to a space, then squeezes sequences of spaces in the output file.

infile: -Name\t\t\t-Employee Number\t\t\t-Date\t-Check Number

outfile: -Name -Employee Number -Date -Check Number

In this example, the *tr* command reduced all multiple tabs to a single tab, converted all single tabs into a space, then made any multiple spaces, including groups of spaces already in the input file, into a single space.

EXAMPLE 8

Syntax: tr -cs String1 String2.

Purpose: squeeze all but String1 from input, translate, and squeeze String2 from output.

Example: tr -cs [:digit:] [0*] < infile > outfile.

Result: squeezes multiples of any character other than the digits 0-9 from the input file to a single occurrence, then translates each non-digit occurrence to a 0, then squeezes multiple 0s to a single occurrence and writes results to the output file.

infile: aaaa1230000456bbbb

outfile: 012304560

In this example, the *tr* command squeezed the 'a's and 'b's to a single character, then translated the character to a 0, then squeezed multiple 0s, including those already in the input file, to a single 0.

EXAMPLE 9

Syntax: tr -ds String1 String2.

Purpose: delete String1 from input, and squeeze String2 from output.

Example: tr -ds 'a' 'b' < infile > outfile.

Result: deletes all ‘a’s from the input file, then squeezes all sequential occurrences of ‘b’s to one occurrence, and writes the results to the output file.

infile: aaa1bbb2aa3

outfile: 1b23

EXAMPLE 10

Syntax: tr -cds String1 String2.

Purpose: delete all but String1 from input, and squeeze String2 from output.

Example: tr -cds [:digit:] '7' < infile > outfile.

Result: deletes all letters from the input file, then squeezes all sequential occurrences of 7s to one occurrence, and writes the results to the output file.

infile: abc555def777ghi999jkl

outfile: 5557999

Note: even though both String1 and String2 are specified in this and in the previous example, there is no substitution performed. The delete of String1 and squeeze of String2 are two distinct operations performed in one command.

SOME EXERCISES

Here are exercises to allow you to try some of the things you have learned in this article. Each step of the exercises will build on the file created in the previous step. Some steps will ask you to enter the *cat* command against an output file. The *cat* command tells the processor to concatenate the contents of the file to the screen so you can view the results.

SET UP

Prepare a file called exerfile containing the following text:

```
Name           Serial     Hours worked

\Sander A.     \49683     \ØØØØØØ39.5
\Joseph J.     \5Ø912     \ØØØØØØ35.Ø
\Martin C.     \453Ø9     \ØØØØØØ39.5
\Wilson M.     \2126Ø     \ØØØØØØ37.Ø
```

EXERCISE 1 – CHARACTER DELETION

Step 1: enter the following command:

```
tr -d '\\' < exerfile > outfile1
```

Step 2: now if you enter *cat outfile1* you will see that all the backslashes have been deleted in the output file. Note that you needed to 'escape' the backslash for the *tr* command to properly identify the characters upon which to work.

Contents of outfile1:

```
Name          Serial    Hours worked

Sander A.     49683     ØØØØØØ39.5
Joseph J.     5Ø912     ØØØØØØ35.Ø
Martin C.     453Ø9     ØØØØØØ39.5
Wilson M.     2126Ø     ØØØØØØ37.Ø
```

Step 3: next, enter the following command:

```
tr -d '.' < outfile1 > outfile2
```

Step 4: now if you enter *cat outfile2* you will see that all the period (full stop) characters have been deleted.

Contents of outfile2:

```
Name         Serial    Hours worked

Sander A     49683     ØØØØØØ395
Joseph J     5Ø912     ØØØØØØ35Ø
Martin C     453Ø9     ØØØØØØ395
Wilson M     2126Ø     ØØØØØØ37Ø
```

EXERCISE 2 – SEQUENCE REMOVAL

Step 1: enter the following command:

```
tr -s 'Ø' < outfile2 > outfile3
```

Step 2: now if you enter *cat outfile3* you will see that all sequences of zeros have been reduced to one per line.

Contents of outfile3:

```
Name         Serial   Hours worked

Sander A     49683    Ø395
Joseph J     5Ø912    Ø35Ø
Martin C     453Ø9    Ø395
Wilson M     2126Ø    Ø37Ø
```

Step 3: next, enter the following command:

```
tr -s [:blank:] < outfile3 > outfile4
```

Step 4: now if you enter *cat outfile4* you will see that all sequences of the space character have been reduced to one for each group per line.

Contents of outfile4:

```
Name        Serial  Hours worked

Sander A    49683   Ø395
Joseph J    5Ø912   Ø35Ø
Martin C    453Ø9   Ø395
Wilson M    2126Ø   Ø37Ø
```

EXERCISE 3 – ONE-FOR-ONE TRANSLATION

Step 1: enter the following command:

```
tr a-z A-Z < outfile4 > outfile5
```

Step 2: now if you enter *cat outfile5* you will see that all lowercase letters have been translated to uppercase.

Contents of outfile5:

```
NAME        SERIAL  HOURS WORKED

SANDER A    49683   Ø395
JOSEPH J    5Ø912   Ø35Ø
MARTIN C    453Ø9   Ø395
WILSON M    2126Ø   Ø37Ø
```

Step 3: next, enter the following command:

```
tr [:digit:] [#*] < outfile5 > outfile6
```

Step 4: now if you enter *cat outfile6* you will see that all numbers have been changed to the hash character. Remember that the asterisk tells the *tr* command to extend the hash character to the length of the digit character set – in this case, ten.

Contents of outfile6:

```
NAME      SERIAL HOURS WORKED

SANDER A  #####  ####
JOSEPH J  #####  ####
MARTIN C  #####  ####
WILSON M  #####  ####
```

David Chakmakian
Programmer (USA)

AIX news

IBM has announced a new and improved AIX Bonus Pack. New third-party software includes System V Commands and Tools for AIX, Acrobat Reader 4.05, Netscape Communicator 4.75, Chili!Soft ASP, Geodesic Systems' Great Circle Version, GraphOn GlobalHost and GO-Joe, and SCO Tarantellae evaluation software.

For further information contact your local IBM representative.
URL: http://www.ibm.com.

* * *

InstallShield has announced three new InstallShield Multi-Platform Edition products (formerly InstallShield Java Edition) targetting IBM and other platforms and resulting from a joint development programme with IBM.

The new products provide developers with the capability to create one application installation that meets the needs of multiple platforms including AIX, Solaris (SPARC and x86), Linux (Red Hat, Caldera OpenLinux, SuSE Linux, TurboLinux), OS/2, and Windows.

The new Platform Packs build on the platform-specific capabilities found in each operating system, combining both specialized Java classes and native code.

What all this means, apparently, is that Java applications launch correctly without requiring end users to run a separate batch file to launch the JVM, native desktop icons are built, and applications uninstall correctly using the operating system's native control panel.

Also, file associations and properties are set for each native environment, native installation technologies can be incorporated into the installation setup, and software objects are defined and managed so that multiple products can share components.

The Express version uses a file tree interface and 52 ready-made components: 36 different wizard panels and actions and 16 pre-built product components. The Professional edition adds an open API and the ability to incorporate any pre-written code, customized panels, or custom actions into installations.

For further information contact: your local InstallShield Software, 900 National Parkway, Suite 125, Schaumburg, IL 60173-5108, USA.
Tel: (847) 240 9111.
URL: http://www.installshield.com/iemp/.

* * *

Progress Software has begun shipping Version 3.0 of SonicMQ, its messaging server for transporting data over the Internet. Among the new bits is DRA (Dynamic Routing Architecture), which provides the basis for the product's improvements.

It's certified for AIX, HP-UX, Linux, Solaris, and Windows NT.

For further information contact:
rogress Software, 14 Oak Park, Bedford, MA 01730, USA.
Tel: (800) 477 6473.
URL: http://www.progress.com/sonicmq/product_info/index.htm.

xephon