Title: Successful Business Continuity – Part 5 of 5

This is the fifth in a series of articles discussing how to implement AIX in an environment dedicated to business continuity. This article discusses a number of topics which include the automated generation of documentation, job scheduling, providing privileged access to non-administrators, and finally integrating a business continuity mentality into everyday activities of an organization.

This article is the last in the series, the entire series of articles has discussed the following topics:

- Article 1:
  - User Names and UID Numbers
  - Group Names and GID Numbers

- Article 2:
  - Machine names
  - Hostnames
  - Boot adapter and service names
  - Resource group names
  - Aliases

- Article 3:
  - Volume Groups
  - Major Numbers
  - Logical Volumes
  - JFS Log Logical volume names
  - Mount points

- Article 4:
  - MQ Series Queue names and aliases
  - Resource Group start/stop scripts
  - Error logging
  - Error Notification

- Article 5:
  - Automated Documentation
  - Console Access
  - Job Scheduling

- Project Planning

Automated Documentation

Any business continuity or disaster recovery plan is dependent upon accurate and up-to-date documentation. Most problems that are encountered during a disaster recovery implementation are due to documentation not being updated to reflect the system requirements. The solution to this problem is to institute policies and procedures that require all system documentation be updated to reflect changes, however this is much easier said than done. Part of the difficulty is that writing documentation is not fun, and most system administrators would normally perform documentation updates as a last step in a project, unless a more urgent issue arose. Unfortunately more urgent issues always arise, and documentation updates are put on hold until time can be spared for this activity. The result is that documentation maintained under this methodology is almost always out-of-date. The only way to ensure documentation is kept up-to-date is to automate the process of generating it.

Much of the work of monitoring, supporting, and maintaining AIX systems is performed by shell or perl scripts, written by the system administrators. Maintaining current documentation for these scripts is usually performed as a separate process from writing or updating the scripts. I have found that combining the processes of script writing and script documentation can eliminate the time lag between the time the scripts are written or updated, and the time the documentation is created or updated. The reason is usually because system administrators enjoy writing scripts, but despise writing documentation. By combining the two an IT manager can increase the likelihood of successful business continuity, in the event of a disaster.

The methodology of combining script writing with script documentation is to implement policies, guidelines, standards, and procedures for writing scripts which requires the documentation should be embedded within the script, at the time it is written or modified. The activity of script writing, which system administrators usually enjoy, simply includes the documentation activity. Documentation is no longer a drudgery, it is a part of writing scripts. The following shell script template illustrates how to define a generic shell script that includes embedded documentation:

```
#!/usr/bin/ksh93
############################################################
function usagemsg_your_function {
  print "
```

```
Program: your_function

Place a brief description ( < 255 chars ) of your shell
function here.

Usage: ${1##*/} [-?vV]

  Where:
    -v = Verbose mode - displays your_function function info
    -V = Very Verbose Mode - debug output displayed
    -? = Help - display this message

Author: Your Name (YourEmail@address.com)
\"AutoContent\" enabled
"
}
#################################################################
####
#### Description:
####
#### Place a full text description of your shell function here.
####
#### Assumptions:
####
#### Provide a list of assumptions your shell function makes,
#### with a description of each assumption.
####
#### Dependencies:
####
#### Provide a list of dependencies your shell function has,
#### with a description of each dependency.
####
#### Products:
####
#### Provide a list of output your shell function produces,
#### with a description of each product.
####
#### Configured Usage:
####
#### Describe how your shell function should be used.
####
#### Details:
####
#### Place nothing here, the details are your shell function.
####
}
#################################################################
function your_function {
  typeset VERSION="1.0"
  typeset TRUE="1"
  typeset FALSE="0"
  typeset VERBOSE="${FALSE}"
  typeset VERYVERB="${FALSE}"

  while getopts ":vV" OPTION
```



```
  do
      case "${OPTION}" in
          'v') VERBOSE="${TRUE}";;
          'V') VERYVERB="${TRUE}";;
        [?:#]) usagemsg_your_function "${0}" && return 1 ;;
      esac
  done

  shift $(( ${OPTIND} - 1 ))

  trap "usagemsg_your_function ${0}" EXIT

#### Place any command line option error checking statements
#### here.  If an error is detected, print a message to
#### standard error, and return from this function with a
#### non-zero return code.  The "trap" statement will cause
#### the "usagemsg" to be displayed.

  trap "-" EXIT

  (( VERYVERB == TRUE )) && set -x
  (( VERBOSE  == TRUE )) && print -u 2 "# Version........: ${VERSION}"

################################################################

####
#### Your shell function should perform it's specfic work here.
#### All work performed by your shell function should be coded
#### within this section of the function.  This does not mean that
#### your function should be called from here, it means the shell
#### code that performs the work of your function should be
#### incorporated into the body of this function.  This should
#### become your function.
####

  return 0
}
################################################################

your_function “${@}”
```

As can be seen in the example shell script template, it contains a function to generate a usage message for the user. This should be a part of all scripts in some form or fashion and is used as part of the automated documentation process. All parts of this particular shell script template are written as functions, to facilitate their inclusion in a function library. The template includes sections for a description of the scripts purpose, the assumptions, dependencies, products, and usage. Notice each line of the embedded documentation in the template begins with four (4) hash marks (#)

followed by a space character. This identifier is used to indicate which lines of the script are to be extracted by a documentation generator.

There are many documentation generators available, probably the best known is "Doxygen" (http://www.doxygen.org), however they can be complex, difficult to implement for existing scripts, and limited in their portability. For the purpose of generating documentation from shell or perl scripts, it is usually best to write a script that performs this function according to your particular needs and requirements. An example documentation generator, called "autocontent", follows. This example uses a predefined configuration file to identify scripts on local or remote systems, copy the scripts, then extract and build HTML based documentation from them. The "rcp" command is used in this example to copy remote files, but can be easily changed to "scp" if so desired. The documentation for the "autocontent" script is, of course, embedded within the script, and can be automatically generated by configuring and running "autocontent" to extract it's own documentation.

```
#!/usr/bin/ksh93
################################################################
function usagemsg {
  print ""
  print "Program: autocontent"
  print ""
  print "This utility parses a list of scripts, extracts the comments"
  print "from within the script, and builds HTML snippets from the"
  print "extracted comments and source code."
  print "It uses a data file to specify the machine and full path"
  print "file name of the source file. Also specified in the data"
  print "file is the machine and directory location where the"
  print "HTML snippets will be placed."
  print ""
  print ""
  print "Usage: ${1##*/} {-d|-l|-f datafile} [-v] [-o] [-c|-u] [-?]"
  print ""
  print "    Where -d = Use ./AutoContent.dat as the data file"
  print "          -l = Use /usr/local/AutoContent/AutoContent.dat as the data file"
  print "          -f datafile = Specify the AutoContent Data File"
  print "          -v = Verbose Mode"
  print "          -o = Send all HTML output to STDOUT"
  print "          -c = Generate code document only"
  print "          -u = Generate usage document only"
  print "          -? = Display usage and help message"
  print ""
  print "Author: Dana French (dfrench@mtxia.com)"
  print ""
  print "\"AutoContent\" enabled"
  print ""
}
################################################################
####
#### Description:
####
#### This utility parses a list of scripts, extracts the comments
#### from within the script, and builds HTML snippets from the
#### extracted comments and source code.  The HTML snippet is saved
#### in a location designated by the user under a file name comprised
#### of the original file name suffix up to the first dot ".", followed
#### by ".content.shtml".  The "shtml" extention is used because
```

```
#### the HTML snippet is intended to be used as part of a server-side
#### include document on a web server.
####
#### Operation of "AutoContent" is controlled by a data file that
#### contains information about the files to be processed.  Each
#### line of the data file represents a record of information and
#### is processed in sequence.  Each line of data should be
#### formatted into a "source" and "destination" portion.  The
#### source portion designates a file to be processed by
#### "AutoContent" and the machine on which it resides.  The
#### destination portion of the data line designates a machine
#### and directory location to place the results of the processing.
####
#### The data file consists of a series of lines, one record
#### per line.  A record consists of several fields
#### specifying the source machine, full path file name,
#### followed by an optional field whose contents are
#### variable.  The next field specifies the destination machine
#### and should be separated from the previous set of "source"
#### fields using a space, tab, pipe symbol or comma.  Following
#### the destination machine name should be the full path
#### destination directory name.
####
#### Example data file records:
####
#### xyzmach:/usr/local/bin/script01.ksh:-?  webserver:/www/httpd/html/scripts
####
#### xyzmach:/usr/local/bin/script02.ksh|webserver:/www/httpd/html/scripts
####
#### xyzmach:/usr/local/bin/script03.ksh:-?,webserver:/www/httpd/html/scripts
####
#### In the above examples, the optional field following the
#### full path source file name contains the characters "-?".
#### This causes "AutoContent" to execute this script with a
#### "-?" option on the command line expecting to receive a
#### usage message.  The usage message is captured and added
#### to the documentation.
####
#### Assumptions:
####
#### It is assumed that any file defined in the data file with
#### the "-?" optional field, is an executable file, recognizes
#### the "-?" option and generates a usage message if the script is
#### executed with that option.  Each file defined using
#### the "-?" optional field WILL BE EXECUTED with the
#### "-?" option to generate the usage message.  If the script does not
#### recognize the "-?" option, THE SCRIPT WILL EXECUTE as though no
#### command line arguments were provided and perform whatever tasks
#### it does under that condition.  Be sure that any file referenced
#### by "autocontent" using the "-?" optional field, recognizes
#### the "-?" option.
####
#### The "autocontent" script generates its list of files for which
#### it generates documentation from the files contained in
#### the "/usr/local/AutoContent" directory.  Scripts should NOT be
#### stored in this directory, only a symbolic link to the script
#### should exist in "/usr/local/AutoContent".
####
#### Additional documentation may be generated if the comments within
#### the script conform to the "autocontent" technique of imbedding
#### comments in scripts.
####
#### Dependencies:
####
#### The list of scripts for the HTML snippet documents are generated, is
#### embedded within the "autocontent" script.  To change the list, the
#### "autocontent" script must be edited.
####
#### The "autocontent" script is a Korn Shell 93 script and must
#### be executed using a Korn Shell 93 compliant script interpreter.
```

```
####
#### Products:
####
#### For each specified script, the "autocontent" script generates
#### an HTML snippet file, that contains the usage message and any
#### additional comments extracted from the script.  Also produced
#### is a separate HTML snippet file that contains the script itself
#### enclosed in HTML tags to preserve formatting.
####
#### Configured Usage:
####
#### This script requires no arguments and can be run from the command
#### line, scheduled, or executed from within another script.
#### This script does not perform any file transfers.  How the
#### files generated by this script are utilized is beyond the scope
#### of this script.
####
#### Details:
####
#####################################################################
TRUE=1
FALSE=0
VERBOSE="${FALSE}"
STDOUT="${FALSE}"
USAGEDOC="${TRUE}"
CODEDOC="${TRUE}"
export DD_TMP="${DD_TMP:-/tmp}"
TMPFILE="/tmp/tmp${$}.tmp"

while getopts ":vdlf:ocu" OPTION
do
    case "${OPTION}" in
        'd') AUTOCONTENT="./AutoContent.dat";;
        'l') AUTOCONTENT="/usr/local/AutoContent/AutoContent.dat";;
        'f') AUTOCONTENT="${OPTARG}";;
        'o') STDOUT="${TRUE}";;
        'c') CODEDOC="${TRUE}"
             USAGEDOC="${FALSE}";;
        'u') USAGEDOC="${TRUE}"
             CODEDOC="${FALSE}";;
        'v') VERBOSE="${TRUE}";;
        '?') usagemsg "${0}" && exit 1 ;;
    esac
done

shift $(( ${OPTIND} - 1 ))

trap "usagemsg ${0}" EXIT
DATAFILE="${AUTOCONTENT:?ERROR: run \"${0} -?\" for help and usage}"
trap "-" EXIT

if [[ -f "${AUTOCONTENT}" ]]
then
   (( VERBOSE == TRUE )) && print -u2 "# Specified data file found"
   (( VERBOSE == TRUE )) && print -u2 "#     ${AUTOCONTENT}"
else
    print -u2 "# AutoContent data file does not exist"
    print -u2 "#     ${AUTOCONTENT}"
    exit 2
fi

#####################################################################
####
#### This script is primarily used to automatically generate documentation
#### content in association with the Disaster Recovery scripts.  However
#### this is an arbitrary association, this script is generic and can
#### be used with any scripts which conform to the rules by which
#### "AutoContent" operates.
####
#### Data lines are read from a user designated file that contains
```

```
#### information that controls the operation of "AutoContent".
#### Each line of data should be formatted into a "source" and
#### "destination" portion.  The source portion designates a file
#### to be processed by "AutoContent" and the machine on which it
#### resides.  The destination portion of the data line designates
#### a machine and directory location to place the results of the
#### processing.
####
#### Each line in the data file should be formatted as follows:
####
#### {source machine name}:{Full Path File Name}[:-?]
#### {space, tab, comma or pipe}
#### {destination machine name}:{Full Path Directory Name}
####
#### If the option flag "-?" is used at the end of a source
#### file name, the file will be treated as as script and
#### executed using the option flag as an argument.  This
#### assumes the option flag will cause the script to
#### generate a usage message which will be captured and
#### included in the generated documentation.
####
#### If used, the optional flag must be separated from the
#### source file name using a colon (:).
####
#####################################################################

(( VERBOSE == TRUE )) && print -u2 "# Building scripts overview document"

while IFS="" read -r -- LINE
do
  if [[ "${LINE}" = *+([[:blank:]]|,|\|)* ]]
  then
    (( VERBOSE == TRUE )) && print -u2 "\n# Data line properly divided into SRC and DEST."
  else
    print -u2 "\n# ERROR: Data line improperly formatted."
    print -u2 "# ERROR: Unable to determine SRC and DEST areas."
    print -u2 "# ERROR: ${LINE}\n"
    continue
  fi

#####################################################################
####
#### The source portion of the data line is extracted from
#### the data line by deleting the largest matching pattern from
#### the end of the line that matches anything up to the
#### first space, tab, comma, or pipe symbol in the line.  The
#### result contains the source machine name, the source file name,
#### and possibly an option flag.  The format of the result should
#### have a colon (:) between the source machine name, the source
#### file name, and if present, the option flag.
####
#### If the result is formatted correctly, it is separated in to
#### its components.  If the option flag is present, it is
#### saved in a variable named "SRCFLAG".
####
#####################################################################

  SRCFLAG=""
  SRC="${LINE%%+([[:blank:]]|,|\|)*}"

  if [[ "${SRC}" = *:* &&
        "_${SRC##*:}" != '_' &&
        "_${SRC%%:*}" != '_' &&
        "_${SRC#*:}" != _-?* ]]
  then
    SRCMACH="${SRC%%:*}"
    SRCFILE="${SRC#*:}"
    if [[ "_${SRC##*:}" = _-?* ]]
    then
```

```
    (( VERBOSE == TRUE )) && print -u2 "# SRC portion of data line properly divided into
MACH, FILE, and FLAG."
      SRCFILE="${SRCFILE%%:*}"
      SRCFLAG="${SRC##*:}"
    else
    (( VERBOSE == TRUE )) && print -u2 "# SRC portion of data line properly divided into MACH
and FILE."
    fi
  else
    print -u2 "\n# ERROR: SRC portion of data line improperly formatted."
    print -u2 "# ERROR: Unable to determine MACH and FILE areas."
    print -u2 "# ERROR: ${SRC}\n"
    continue
  fi

#################################################################
####
#### The destination portion of the data line is extracted from
#### the data line by deleting the largest matching pattern from
#### the beginning of the line that matches anything up to the
#### last space, tab, comma, or pipe symbol in the line.  The
#### result contains both the destination machine name and the
#### destination directory.  The format of the result should have
#### a colon (:) between the destination machine name and the
#### destination directory.
####
#### If the result is formatted correctly, it is separated in to
#### its components.
####
#################################################################

  DEST="${LINE##*+([[:blank:]]|,|\|)}"

  if [[ "${DEST}" = *:* &&
        "_${DEST##*:}" != '_' &&
        "_${DEST%:*}" != '_' ]]
  then
    (( VERBOSE == TRUE )) && print -u2 "# DEST portion of data line properly divided into
MACH and DIR."
    DESTMACH="${DEST%%:*}"
    DESTDIR="${DEST#*:}"
  else
    print -u2 "\n# ERROR: DEST portion of data line improperly formatted."
    print -u2 "# ERROR: Unable to determine MACH and DIR areas."
    print -u2 "# ERROR: ${DEST}\n"
    continue
  fi

#################################################################
####
#### Define the command to use to copy the source file from its
#### original location into a temporary file.  Assume the source
#### file exists on a remote machine and define the copy command
#### as a remote copy, followed by the source machine name,
#### followed by a colon.  Then test to see if the source machine
#### name is the same as the local machine name or defined as
#### "localhost".  If so, reset the copy command to be a simple
#### copy followed by a space.
####
#### After defining the copy command, perform the copy.
####
#################################################################
  CMD_CP="rcp ${SRCMACH}:"
  [[ "${SRCMACH}" = *$( uname -n )* ]] && CMD_CP="cp "
  [[ "${SRCMACH}" = localhost ]] && CMD_CP="cp "
  rm -f "${TMPFILE}"
  ${CMD_CP}${SRCFILE} ${TMPFILE}

  FILENAME="${SRCFILE##*/}"
```

```
#################################################################
####
#### For each script, an HTML snippet file is created to contain
#### the usage message and any additional "autocontent" compliant
#### comments that can be extracted from the script.  This file is
#### named using the file name suffix of the original script up to
#### but not including the first dot ".", followed by "doc.content.shtml".
####
#### The ".shtml" is used so the document may additionally use
#### server-side includes.
####
#################################################################
  if (( USAGEDOC == TRUE ))
  then

    (( VERBOSE == TRUE )) && print -u2 "# Building ${FILENAME} usage document"

    OUTFILE="${DD_TMP}/${FILENAME%%.*}doc.content.shtml"
    (( STDOUT == TRUE )) && OUTFILE='&1'

    eval "exec 3>${OUTFILE}"

    print -u3 "<!-- Begin \"${FILENAME}doc.content.shtml\" -->"

#################################################################
####
#### Each Script is executed with the "-?" option to generate the
#### usage message associated with the script.  This usage message
#### is saved in the documentation for the script.
####
#### Any "<" or ">" symbols generated by the usage message or extracted
#### from the script in the additional comments, are converted to
#### HTML recognizable codes that will be interpreted by the web
#### browser when the page is displayed.
####
#################################################################

    if [[ "_${SRCFLAG}" != "_" ]]
    then
      (( VERBOSE == TRUE )) && print -u2 "# Executing script to generate usage message"
        print -u3 "<P><BLOCKQUOTE><STRONG><PRE><CODE>"
        chmod 755 "${TMPFILE}"
        /usr/bin/ksh93 "${TMPFILE}" -? |
          sed -e "s/</\&lt;/g;s/>/\&gt;/g;s|${TMPFILE##*/}|${SRCFILE##*/}|g" |
          grep -v "ERROR" >&3

        print -u3 "</CODE></PRE></STRONG></BLOCKQUOTE></P>"
    fi

    (( VERBOSE == TRUE )) && print -u2 "# Generating additional documentation for
\"${FILENAME}\""
    print -u3 "<P><HR></P>"

#################################################################
####
#### Additional comments may be extracted from the scripts if the
#### comments conform to the "autocontent" technique of commenting
#### scripts.  This technique extracts only those comments embedded
#### within a script which begin with four hash marks followd by
#### a single space (#### ).  This pattern must also occur at the
#### beginning of the line.  Any comments which begin with this
#### pattern are extracted and reformatted as an HTML paragraph.
####
#### Multiple paragraphs may be designated within the script by
#### using the (#### ) pattern with nothing following.  This will
#### be interpreted by the "autocontent" generated to mean "insert
#### end of paragraph tag followed by a begin paragraph tag".
####
#### Any extracted comment line which ends with a colon ":" will
#### be enclosed in HTML STRONG tags to make the text bold when
```

```
#### displayed in a browser.
####
#### If "autocontent" generates multiple "End Paragraph -
#### Begin paragraph" pairs, they will be collapsed into a single pair.
####
#################################################################

    grep "^#### " "${TMPFILE}" |
      sed -e 's/^#### //g;s/^$/<\/P><P>/g' |
      uniq |
      sed -e '1,1 s/<\/P><P>/<P>/g;$,$ s/<\/P><P>/<\/P>/g' |
      sed -e 's/.*:$/<STRONG>&<\/STRONG>/g' >&3
#
# A server-side-include directive that displays the date when
# the document was generated is added to the end of each document.
#
    print -u3 "<P><!--#config timefmt=\"%D\" -->"
    print -u3 "This file last modified <!--#echo var=\"LAST_MODIFIED\" --></P>"

    print -u3 "<!-- End \"${FILENAME}doc.content.shtml\" -->"

    exec 3>&-

  fi    # (( USAGEDOC == TRUE ))

#################################################################
####
#### Also for each script, an HTML snippet file is created to contain
#### the script source code.  This file is
#### named using the file name suffix of the original script up to
#### but not including the first dot ".", followed by ".content.shtml".
####
#### The ".shtml" is used so the document may additionally use
#### server-side includes.
####
#################################################################

  if (( CODEDOC == TRUE ))
  then

    (( VERBOSE == TRUE )) && print -u2 "# Building ${FILENAME} code document"

    OUTFILE="${DD_TMP}/${FILENAME%%.*}.content.shtml"
    (( STDOUT == TRUE )) && OUTFILE='&1'

    eval "exec 3>${OUTFILE}"

    print -u3 "<!-- Begin \"${FILENAME}.content.shtml\" -->"
    print -u3 "<P><H3>Script Source Code for \"${FILENAME}\"</H3></P>"
    print -u3 "<P>This document contains the source code for the"
    print -u3 "Disaster Recovery script \"${FILENAME}\"."
    print -u3 "</P><P><HR></P><P><BLOCKQUOTE><PRE><CODE>"

#################################################################
####
#### Any "<" or ">" symbols generated by the usage message or extracted
#### from the script in the additional comments, are converted to
#### HTML recognizable codes that will be interpreted by the web
#### browser when the page is displayed.
####
#################################################################

    cat "${TMPFILE}" |
      sed -e "s/</\&lt;/g;s/>/\&gt;/g;s/\\\&/\\\\\&amp;/g;" >&3

    print -u3 "</CODE></PRE></BLOCKQUOTE></P>"

#################################################################
####
#### A server-side-include directive that displays the date when
```

```
#### the document was generated is added to the end of each document.
####
################################################################

    print -u3 "<P><!--#config timefmt=\"%D\" -->"
    print -u3 "This file last modified <!--#echo var=\"LAST_MODIFIED\" --></P>"

    print -u3 "<!-- End \"${FILENAME}.content.shtml\" -->"

    exec 3>&-

  fi    # (( CODEDOC == TRUE ))

  rm -f "${TMPFILE}"

  if (( STDOUT == FALSE ))
  then
    CMD_CP="rcp"
    DESTNAME="${DESTMACH}:${DESTDIR}"
    if [[ "${SRCMACH}" = *$( uname -n )* ]]
    then
      DESTNAME="${DESTDIR}"
    fi

    if [[ "${DESTMACH}" = *$( uname -n )* ]]
    then
      DESTNAME="${DESTDIR}"
    fi

    if [[ "${SRCMACH}" = *$( uname -n )* && "${DESTMACH}" = *$( uname -n )* ]]
    then
      CMD_CP="cp"
    fi

    if (( USAGEDOC == TRUE ))
    then
      chmod 644 "${DD_TMP}/${FILENAME%%.*}doc.content.shtml"
      (( VERBOSE == TRUE )) && print -u2 "# Copying usage document file to destination"
      (( VERBOSE == TRUE )) && print -u2 "# Destination: ${DESTNAME}"
      ${CMD_CP} "${DD_TMP}/${FILENAME%%.*}doc.content.shtml" "${DESTNAME}"
      rm -f "${DD_TMP}/${FILENAME%%.*}doc.content.shtml"
    fi    # (( USAGEDOC == TRUE ))

    if (( CODEDOC == TRUE ))
    then
      chmod 644 "${DD_TMP}/${FILENAME%%.*}.content.shtml"
      (( VERBOSE == TRUE )) && print -u2 "# Copying code document file to destination"
      (( VERBOSE == TRUE )) && print -u2 "# Destination: ${DESTNAME}"
      ${CMD_CP} "${DD_TMP}/${FILENAME%%.*}.content.shtml" "${DESTNAME}"
      rm -f "${DD_TMP}/${FILENAME%%.*}.content.shtml"
    fi    # (( CODEDOC == TRUE ))

  fi

done < "${AUTOCONTENT}"
```

Documentation Look-and-Feel Generators

In order to provide flexibility and portability to the automatically generated documentation, it should be generated only as document content, the look-and-feel, formatting, navigation, etc. should be generated separately. This permits the documentation to be manipulated, rearranged, regenerated, or reformatted without disturbing the actual content.

The previously mentioned “Doxygen” normally generates the documentation and the look-and-feel of the pages, which may be desirable under some circumstances, however under other circumstances it may be regarded as limiting. It has the tendency to lock-in the “Doxygen” methodology for managing documentation, and makes it difficult to change when newer, better methods come along. Another drawback of applications such as “Doxygen” is they tend to be somewhat complex and rigid. In this day of web sites and HTML based documentation, it is usually quicker and easier to build a small, simple HTML look-and-feel generator. To generate the documentation in HTML, it needs to be able to create the following parts and pieces for each document:

- Header
- Title
- Footer
- Navigation Links

Usually the most difficult part of this is generating the navigation links, because each document will need to be linked to one or more other documents. Determining how to create these linking structures can be a daunting task. However, applying some simple rules can resolve this issue quite easily.

- Work from the perspective of one document at a time
- Each document can have zero or more parent documents
- Each document can have zero or more child documents.

With these three simple rules, any and all navigation structures can be built. These rules require that each document have a parent document, and if each document has a parent, a structured relationship can be established between all documents. All that is necessary is to keep a parent-child relationship for each document, from this all other relationships can be derived. For example consider the following document names:

grandparent1.content.shtml
grandparent2.content.shtml
grandparent3.content.shtml
grandparent4.content.shtml
parent1.content.shtml
parent2.content.shtml
child.content.shtml

grandchild1.content.shtml
grandchild2.content.shtml

Establishing parent-child relationships for these documents would appear as follows:

Parent Document	Child Document
NULL	grandparent1
NULL	grandparent2
grandparent1	parent1
grandparent2	parent1
grandparent3	parent2
grandparent4	parent2
parent1	child
parent2	child
child	grandchild1
child	grandchild2

From this simple structure it can be derived the document “child” has parent documents named “parent1” and “parent2”. The document “child” also has children “grandchild1” and “grandchild2”. Furthermore, all other document relationships to “child” can be derived from this structure. These relationships can then be used to build navigation links for each document.

An example “Look-and-Feel” generator which uses this parent-child relationship to build navigation links follows:

```
#!/usr/bin/ksh93
##################################################################
function usagemsg_menugen {
  print "
Program: menugen

Create naviation and look-and-feel characteristics for a
directory of web content files.

Usage: menugen [-g]] [-S]] [-u]] [-f datafile]] | -c | -C | -h
     -c = Create a default \"menugen.cfg\" file
     -C = Overwrite \"menugen.cfg\" file, even if it already exists
     -g = Show grandchild links in menus
     -R = Position menu bar on Right side of page
     -S = If the static menu bar exists, overwrite it with the default
     -f = Use data file specified by value \"datafile\"
```

```
     -u = Underline Menu Links
     -n = Generate .html files
     -v = Verbose mode
     -V = Very Verbose mode
     -h = Display help file

Author: Dana French (dfrench@mtxia.com) Copyright 2005
\"Autocontent Enabled\"
"
}
#################################################################
####
#### Description
####
#### "menugen" is a shell script that provides the ability to
#### easily maintain and modify web based documentation.
#### This script creates a standard "look-and-feel" for user
#### supplied HTML based content, and generates navigation
#### links between the pages.  An web server feature called
#### "Server Side Includes" (SSI) is utilized by "menugen" to
#### reference the various parts and pieces of each page.
####
#### To use "menugen", the user should create content and
#### store it in files called "XXXXXXXX.content.shtml", where
#### XXXXXXXX is some prefix file name defined by the user.
#### "menugen" creates menus, links, headers, titles,
#### footers, contact info links, etc, based on information
#### stored in a user defined relationship file.  The
#### relationship file is assumed to be in the same directory
#### with the "XXXXXXXX.content.shtml" files and is called
#### "menugen.dat".
####
#### "menugen" allows users to modify the "look-and-feel" of
#### all files in a directory by storing configuration
#### information such as colors, font sizes, title info, etc
#### in a file called "menugen.cfg".
####
#### The "-c" option will create a file called "menugen.cfg"
#### if it does not already exist.  If it does exist it will
#### not overwrite it.  To recreate the "menugen.cfg" file,
#### you will first have to remove it.  The "menugen.cfg"
#### file contains user modifiable parameters such as font
#### colors, font sizes background color, title and contact
#### information.
####
#### One of the values defined in the "menugen.cfg" file is
#### called "THEME".  This is a title which is displayed at
#### the top of every page.  The title can be a two part
#### title separated by a colon (:).  The part of the title
#### to the left of the colon will be displayed in a larger
#### text.  The part of the title to the right of the colon
#### will be displayed in italics and a smaller text.  All
#### values in the configuration file may be changed to suit
#### the users particular needs.  The "menugen.cfg" file MUST
#### be executeable.
####
#### The "menugen" program creates a menu file for each page
#### which contains the relationships between each page, its
#### parent page, and its child pages.  The links displayed
#### on the menu by default are that of the parent and
#### children of each page.  The "-g" option causes the
#### grandchildren of each page to also be included in the
#### menu.
####
#### Normally, if the static menu bar file exists, it is not
#### overwritten in order to preserve any user customization
#### of that file.  However the "-S" option will cause the
#### static menubar file to be overwritten with the default
#### values.  The static menu bar file is "staticbar.shtml".
####
```

```
#### The datafile describes the relationships between all the
#### web pages in the current directory.  The relationship
#### between the pages is maintained via parent/child
#### references.  By default the relationships are read from
#### a file called "menugen.dat" in the current directory.
#### The "-f" option allows the user to specify a different
#### file name.  The datafile has the following format;
####
#### PARENT<tab>CURRENT<tab>SHORT DESCRIPTION
####
#### PARENT and CURRENT are file name prefixes and are
#### usually limited to 8 characters or less.  TAB characters
#### between the fields is mandatory.  PARENT is the parent
#### of the CURRENT page.  The SHORT DESCRIPTION is a
#### description of the CURRENT page.  An example of datafile
#### would be;
####
#### NULL             index          Home Page
#### index            child1         Child 1 of index
#### index            child2         Child 2 of index
#### index            child3         Child 3 of index
#### child1           child11        Child 1 of child1
#### child1           child12        Child 2 of child1
#### child1           child13        Child 3 of child1
#### child1           child14        Child 4 of child1
#### child2           child21        Child 1 of child2
#### child2           child22        Child 2 of child2
#### child2           child23        Child 3 of child2
#### child3           child31        Child 1 of child3
#### child3           child32        Child 2 of child3
####
#### In this example, the page "index" has three child pages
#### called "child1", "child2", and "child3".  The menu on
#### the "index" page would reference these three pages.  The
#### page "child1" has four child pages called "child11",
#### "child12", "child13", "child14".  The menu on the
#### "child1" page would reference these four pages.  And so
#### on for child pages "child2" and "child3".
####
#### The "-h" option displays this help file and is assumed
#### to reside in the file called
#### "/usr/local/sh/README.menugen".
####
#### Most graphical web browsers will show hypertext links as
#### underlined text of a different color than normal text.
#### When this program creates the menu bars, the links are
#### configured NOT to have underlines.  If you want
####
#### the links in the menu bars to have underlines, use the
#### "-u" option.
####
#### Assumptions:
####
#### It is assumed that a file exists which describes a
#### parent-child relationship for each document.  The format
#### of this file assumes the first column contains the
#### prefix name of the parent document, the second column
#### contains the prefix name of the child document, and the
#### third column contains a very short description (16 char or
#### less), of the child document.
####
#### The "content" files are assumed to be in the same
#### directory as the "menugen.dat" file.
####
#### Dependencies:
####
#### This script requires a "menugen.dat" file be created by
#### the user, which is normally a manual process, however
#### can be automated depending upon desired usage.
####
```

```
#### Products:
####
#### This script generates various parts and pieces
#### comprising the look-and-feel of the web pages.  This
#### includes a title (title.shtml), header (X.shtml), footer
#### (footer.shtml), dynamic navigation links (X.menu.shtml),
#### static navigation links (staticbar.shtml), and a
#### configuration file (menugen.cfg).  "X" in the above file
#### names represents the filename prefix of each page
#### identified in the "menugen.dat" file.
####
#### Configured Usage:
####
#### This script is written to be used as a command line
#### program, but can be scheduled to run through cron or any
#### other scheduling system.  If scheduled, the current
#### directory must be the same as the directory containing
#### the "menugen.dat" file.
####
#### Details:
#################################################################
#################################################################
####
####
#### The "mkconfig" function checks for or creates a
#### configuration file for the "menugen" program.  This
#### configuration file contains several parameter settings
#### that control text, color, font, sizes, etc.
####
#### If the "menugen" configuration file already exists, do not
#### overwrite it, simply return with a non-zero exit code.
#### If the "menugen" configuration file does not exist,
#### create it and exit with a zero exit code.
####
#################################################################
mkconfig()
{
  if [[ -f "./menugen.cfg" ]]
  then
      print "\"menugen.cfg\" file already exists, not overwritten"
      return 1
  fi
  print "THEME=\"Default Theme: Default Subtheme\"       # Title displayed at top of every page" >
./menugen.cfg
  print "CLR_TEXT=\"#000000\"            # Text color (default=#000000)" >> ./menugen.cfg
  print "CLR_BACKGRD=\"#FFFFFF\"         # Background color (default=#FFFFFF)" >> ./menugen.cfg
  print "CLR_LINK=\"#0000FF\"            # Link color (default=#0000FF)" >> ./menugen.cfg
  print "CLR_ALINK=\"#777700\"           # Active Link color (default=#777700)" >> ./menugen.cfg
  print "CLR_VLINK=\"#770000\"           # Viewed Link color (default=#770000)" >> ./menugen.cfg
  print "CLR_MENUBORDER=\"#ABADC9\"      # Menu Border color (default=#ABADC9)" >> ./menugen.cfg
  print "CLR_MENUBG=\"#DDDDDD\"          # Menu Background color (default=#DDDDDD)" >>
./menugen.cfg
  print "CLR_MENUTEXT=\"#000000\"                # Menu Text color (default=#000000)" >>
./menugen.cfg
  print "SIZ_BASEFONT=\"4\"              # Base Font Size (default=4)" >> ./menugen.cfg
  print "CONTACT_NAME=\"Dana French\"    # Person responsible for this Page (default=Dana French)"
>> ./menugen.cfg
  print "CONTACT_EMAIL=\"dfrench@mtxia.com\"     # Email address of person responsible for this
page (default=dfrench@mtxia.com)" >> ./menugen.cfg
  chmod 755 ./menugen.cfg
  print "\"menugen.cfg\" file was created in the current directory."
  return 0
}
#################################################################
####
#### The "mg_footer" function outputs the "footer" section of
#### the HTML pages, containing the information from the
#### "configuration" file.
####
#################################################################
```

```
mg_footer()
{
  print "
<!-- Begin included file \"footer.shtml\" -->
  <P><!--#include file=\"hr.shtml\" --></P>

  <P>
  <FONT Size=\"2\">
  <ADDRESS>
  For information regarding this page, contact
  <A Href=\"mailto:${CONTACT_EMAIL}\">
  ${CONTACT_NAME} ( ${CONTACT_EMAIL} )</A>
  </ADDRESS>
  </FONT>
  </P>

<!-- End included file \"footer.shtml\" -->
"
}
#####################################################################
####
#### The "mg_hr" function outputs the "horizontal rule"
#### section of the HTML pages, using the parameter settings
#### from the "configuration" file.
####
#####################################################################
mg_hr()
{
  print "
<!-- Begin included horizontal rule file \"hr.shtml\" -->

  <TABLE Border=\"0\" Width=\"100%\" Cellspacing=\"0\" Cellpadding=\"0\">
  <TR><TD Bgcolor=\"${CLR_MENUBORDER}\"><FONT Size=\"1\"> </FONT></TD></TR></TABLE>

<!-- End included horizontal rule file \"hr.shtml\" -->
"
}
#####################################################################
####
#### The "mg_staticbar" function outputs the "static menu
#### bar" section of the HTML pages, using the parameter
#### settings from the "configuration" file.  Currently this
#### section appears at the top of each HTML page.
####
#####################################################################
mg_staticbar()
{
  print "
<!-- Begin included static linkbar file \"staticbar.shtml\" -->
"

####
#### If the "underline" option was specified on the command
#### line, output a style tag to cause HTML links in the static
#### menu bar to be underlined.
####

  if (( UNDERLINE == TRUE ))
  then
      print "
  <STYLE>
  <!--
        A { Color:${CLR_MENUTEXT}; text-decoration:none; }
        A:hover { text-decoration:underline;color:${CLR_MENUTEXT}; }
  //-->
  </STYLE>
"
  fi

####
```

```
#### Output the "static menu bar" containing a static set of
#### HTML links.  This program generates a hardcoded set of
#### links and does not currently read these from the
#### configuration file.  This will be a future enhancement.
####

  print "
  <P>
  <TABLE Border=\"0\" Cellspacing=\"5\" Cellpadding=\"0\" Align=\"right\"><TR><TD
Bgcolor=\"white\">
  <TABLE Border=\"0\" Width=\"100%\" Cellspacing=\"0\" Cellpadding=\"1\" Align=\"left\"><TR><TD
Bgcolor=\"${CLR_MENUBORDER}\">
  <TABLE Border=\"0\" Width=\"100%\" Cellspacing=\"0\" Cellpadding=\"5\" Align=\"left\">
  <P>
    <TR>
      <TD Bgcolor=\"${CLR_MENUBG}\" Align=\"left\" Valign=\"top\">
        <FONT Size=\"2\" Color=\"${CLR_MENUTEXT}\"><STRONG>
  <A Href=\"http://www.hcahealthcare.com\"> HCA</A>
   | <A Href=\"http://www.teamggi.com\"> GGI</A>
   | <A Href=\"http://www.mtxia.com/\"> Mt Xia</A>
        </STRONG></FONT>
      </TD>
    </TR>
  </P>
  </TABLE>
  </TD></TR></TABLE>
  </TD></TR></TABLE>
  </P>
"

####
#### If the "underline" option was specified on the command
#### line, output a style tag to cause HTML links in the menu
#### bar to be turned off.
####

  if (( UNDERLINE == TRUE ))
  then
      print "
  <STYLE>
  <!--
        A { Color:${CLR_LINK}; text-decoration:underline; }
        A:hover { text-decoration:underline;color:${CLR_LINK}; }
  //-->
  </STYLE>
"
  fi

  print "
<!-- End included static linkbar file \"staticbar.shtml\" -->
"
}
#################################################################
####
#### The "mg_titlebar" function outputs the "title bar"
#### section of the HTML pages, using parameter settings from
#### the "configuration" file. If the "THEME" parameter
#### contains a colon character (:), the value is split into
#### two parts using the colon character as a delimiter. The
#### first part of the THEME value is displayed in a larger
#### text than the second part.
####
#################################################################
mg_titlebar()
{
  THEME1="${THEME}"
  THEME2=" "
  if [[ "_${THEME}" == _*:* ]]
  then
      THEME1="${THEME%%:*}"
```

```
      THEME2="${THEME#*:}"
  fi

  print "
<!-- Begin included file \"titlebar.shtml\" -->

  <P><FONT Size=\"6\" Color=\"#000000\">
  <BR><STRONG>${THEME1}:</STRONG></FONT>
  <FONT Size=\"4\" Color=\"#000000\"><STRONG><I>${THEME2}</I></STRONG></FONT>
  <BR><!--#include file=\"hr.shtml\" --></P>

<!-- End included file \"titlebar.shtml\" -->
"
}
#####################################################################
####
#### The "cat_file" function reads a file specifed by the
#### first command line argument to this function, and
#### outputs the content of the file to standard output.
#### This is similar to the Unix "cat" command.  the purpose
#### of this function is to avoid executing the external Unix
#### command "cat".
####
#####################################################################
cat_file()
{
  if [[ -s "${1}" ]]
  then
    while IFS=$'\n' read -r -- DATALINE
    do
        print -r -- "${DATALINE}"
    done < "${1}"
  else
    print -u 2 "#   WARNING: ${1} does not exist"
  fi
}
#####################################################################
####
#### The "menugen" function is the main portion of this
#### program.  It generates the various parts and pieces of
#### each web page based on a parent-child relationship
#### specified in a datafile.  This datafile contains the
#### file name prefix of each file to be processed by this
#### program.
####
#####################################################################
function menugen {

  VERSION="10.0"
  TRUE="1"
  FALSE="0"
  EXITCODE="0"
  VERBOSE="${FALSE}"
  VERYVERB="${FALSE}"
  DATAFILE="./menugen.dat"
  GRANDCHILD="${FALSE}"
  HTMLFILES="${FALSE}"
  UNDERLINE="${FALSE}"
  MENUPOS="left"

#####################################################################
####
#### Check for the existance of the default configuration
#### file and check to see that it is executable.  If it is
#### not executable, make it so.  The default configuration
#### file is always assumed to be in the current directory.
#### After checking the existance and execute bit, execute
#### the default configuration file for the purpose of
#### establishing several shell variables that define
#### look-and-feel parameters for colors, font sizes, etc.
```

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

  if [[ -s "./menugen.cfg" ]]
  then
    if [[ ! -x "./menugen.cfg" ]]
    then
      if ! chmod 755 "./menugen.cfg"
      then
        print "Unable to make \"menugen.cfg\" executable"
        return 6
      fi
    fi
    . ./menugen.cfg
  fi

#################################################################
####
#### For each of the standard parameter values in the default
#### configuration file, check to see if the shell variable
#### is unset or null.  If so, assign the variable a default value.
####
#################################################################

  THEME="${THEME:-'Default Theme: Default Subtheme'}"
  CLR_TEXT="${CLR_TEXT:-'#000000'}"
  CLR_BACKGRD="${CLR_BACKGRD:-'#FFFFFF'}"
  CLR_LINK="${CLR_LINK:-'#0000FF'}"
  CLR_ALINK="${CLR_ALINK:-'#777700'}"
  CLR_VLINK="${CLR_VLINK:-'#770000'}"
  CLR_MENUBORDER="${CLR_MENUBORDER:-'#ABADC9'}"
  CLR_MENUBG="${CLR_MENUBG:-'#DDDDDD'}"
  CLR_MENUTEXT="${CLR_MENUTEXT:-'#000000'}"
  SIZ_BASEFONT="${SIZ_BASEFONT:-'4'}"
  CONTACT_NAME="${CONTACT_NAME:-'Dana French'}"
  CONTACT_EMAIL="${CONTACT_EMAIL:-'dfrench@mtxia.com'}"

#################################################################
####
#### Define the default page data file that contains the
#### parent-child relationships between pages.  Also define
#### several other parameter that control the default
#### behavior of this script.
####
#################################################################

  while getopts ":vVcChgRSf:nu" OPTION
  do
      case "${OPTION}" in
          'c') mkconfig
               return 3;;
          'C') rm -f "./menugen.cfg"; mkconfig
               return 5;;
          'h') usagemsg_menugen | more - "/usr/local/sh/README.menugen"
               return 4;;
          'g') GRANDCHILD="${TRUE}";;
          'R') MENUPOS="right";;
          'S') rm -f "./staticbar.shtml";;
          'f') DATAFILE="${OPTARG}";;
          'n') HTMLFILES="${TRUE}";;
          'u') UNDERLINE="${TRUE}";;
          'v') VERBOSE="${TRUE}";;
          'V') VERYVERB="${TRUE}";;
          '?') usagemsg_menugen "${0}" && return 1 ;;
          ':') usagemsg_menugen "${0}" && return 1 ;;
      esac
  done

  shift $(( ${OPTIND} - 1 ))
```

```
  trap "usagemsg_menugen ${0}" EXIT
      if [[ "_${DATAFILE}" = "_" || ! -s "${DATAFILE}" ]]
      then
          usagemsg_menugen "Invalid data file name"
          return 1
      fi
  trap "-" EXIT

################################################################

  (( VERBOSE == TRUE )) &&
      print "# ${DATAFILE} will be used to define pages and relationships"

  (( VERBOSE == TRUE )) &&
      (( GRANDCHILD == TRUE )) &&
       print "# Grandchild links will be shown on menus"

  (( VERYVERB == TRUE )) && set -x

####
#### Generate the static menu bar, title bar, horizontal
#### rule, and page footer.  Only generate the static menu
#### bar if the file does not already exist.  This is so the
#### static menu bar can be modified without being
#### overwritten each time the "menugen" program is executed.
####

  [[ ! -f "./staticbar.shtml" ]] && mg_staticbar > ./staticbar.shtml

  mg_titlebar > ./titlebar.shtml
  mg_hr > ./hr.shtml
  mg_footer > ./footer.shtml

################################################################

####
#### Read the parent-child relationship datafile and assign the
#### values to shell variables.  The data file should contain
#### three fields, parent file name prefix, child file name
#### prefix, and child file name description.  The three
#### fields should be delimited by "tab" characters.
####

  DATATMP0="/tmp/tmp0${$}.out"
  DATATMP1="/tmp/tmp1${$}.out"
  DATATMP2="/tmp/tmp2${$}.out"

  cat_file "${DATAFILE}" > "${DATATMP0}"

  while IFS=$' \t\n' read -r -- PARENTID1 PAGEID1 PAGENAME1
  do
    (( VERBOSE == TRUE )) && print

####
#### Based on the current value of the child file name
#### prefix, re-read the entire datafile and extract only
#### those lines whose parent file name prefix matches the
#### current child file name prefix.  This provides a list of
#### children of the current child file name prefix.
####

    while IFS=$'\n' read -r -- DATALINE
    do
      [[ "${DATALINE}" == ${PAGEID1}+([$'b\t'])* ]] &&
         print -r -- "${DATALINE}"
    done < "${DATATMP0}" > "${DATATMP1}"

####
#### Based on the current value of the parent file name
#### prefix, re-read the entire datafile and extract the
```

```
#### first data line whose child file name prefix matches the
#### current parent file name prefix.  From this dataline,
#### extract the description of the parent of the current
#### child file name prefix and store this value in a shell
#### variable.
####

    while IFS=$'\n' read -r -- DATALINE
    do
      if [[ "${DATALINE}" == *+([$'\t'])${PARENTID1}+([$'\t'])* ]]
      then
          DATALINE=${DATALINE//+([$'\t'])/:}
          PARENTNAME="${DATALINE##*:}"
          break
      fi
    done < "${DATATMP0}"

    (( VERBOSE == TRUE )) && print "# ${PARENTNAME}"
    typeset -u PGNMTMP="${PAGENAME1}"
    (( VERBOSE == TRUE )) && print "#     ${PGNMTMP}"

####
#### If the current data line begins with a comment character
#### (#), skip it and continue with the next data line.
####

    if [[ "_${PARENTID1}" == _\#* ]]
    then
        continue
    fi

####
#### If the "-R" command line option was specified,  then
#### create an "index.shtml" file that positions the "menu
#### bar" on the right hand side of each HTML page.
#### Otherwise generate an "index.shtml" file with the "menu
#### bar" positioned on the left side of each HTML page. The
#### "index.shtml" file is build assuming that server sides
#### includes (SSI) is implemented on whatever web server
#### will host these files.
####

    if [[ "_${MENUPOS}" = "_right" ]]
    then

      print "
<HTML>
<!-- PARENTID1=\"${PARENTID1}\" -->
<!-- PAGEID1=\"${PAGEID1}\" -->
<!-- PAGENAME1=\"${PAGENAME1}\" -->
<HEAD>
<TITLE>${PAGENAME1} - ${THEME}</TITLE>
</HEAD>
<BODY Bgcolor=\"${CLR_BACKGRD}\" Text=\"${CLR_TEXT}\" Link=\"${CLR_LINK}\" Alink=\"${CLR_ALINK}\"
Vlink=\"${CLR_VLINK}\">
<BASEFONT Size=\"${SIZ_BASEFONT}\">
<!--#include file=\"staticbar.shtml\" -->
<!--#include file=\"titlebar.shtml\" -->
<TABLE Width="100%" Cellspacing=\"0\" Cellpadding=\"0\" Border=\"0\">
<P><TR><TD Width=\"80%\" Height=\"10%\" Valign="top">
<P><H2>${PAGENAME1}</H2></P>
<P><HR></P>
</TD><TD Width="20%" Align=\"right\" Valign=\"top\" rowspan=\"2\">
<!--#include file=\"${PAGEID1##*/}.menu.shtml\" -->
</TD></TR></P>
" > "${PAGEID1/#*\///tmp/}.shtml"

    else

      print "
```

```
<HTML>
<!-- PARENTID1=\"${PARENTID1}\" -->
<!-- PAGEID1=\"${PAGEID1}\" -->
<!-- PAGENAME1=\"${PAGENAME1}\" -->
<HEAD>
<TITLE>${PAGENAME1} - ${THEME}</TITLE>
</HEAD>
<BODY Bgcolor=\"${CLR_BACKGRD}\" Text=\"${CLR_TEXT}\" Link=\"${CLR_LINK}\" Alink=\"${CLR_ALINK}\"
Vlink=\"${CLR_VLINK}\">
<BASEFONT Size=\"${SIZ_BASEFONT}\">
<!--#include file=\"staticbar.shtml\" -->
<!--#include file=\"titlebar.shtml\" -->
<!--#include file=\"${PAGEID1##*/}.menu.shtml\" -->
<P><H2>${PAGENAME1}</H2></P>
<P><HR></P>
" > "${PAGEID1/#*\///tmp/}.shtml"

    fi

#################################################################

####
#### Begin building the "menu bar" file.  This will be unique
#### to each HTML page and is based on the parent-child
#### relationships defined in the datafile.
####

    print "

<!-- Begin included menu file \"${PAGEID1}.menu.shtml\" -->
" > "${PAGEID1/#*\///tmp/}.menu.shtml"

####
#### If the "underline" option was specified on the command
#### line, output a style tag to cause HTML links in the
#### menu bar to be underlined.
####

    if (( UNDERLINE == TRUE ))
    then
      print "
<STYLE>
<!--
        A { Color:${CLR_MENUTEXT}; text-decoration:none; }
        A:hover { text-decoration:underline;color:${CLR_MENUTEXT}; }
//-->
</STYLE>
" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
    fi

####
#### Output the "menu bar" presentation to the menu bar file.
####

    print "
<P Align=\"right\">
<TABLE Border=\"0\" Cellspacing=\"5\" Cellpadding=\"0\" Align=\"left\"><TR><TD Bgcolor=\"white\">
<TABLE Border=\"0\" Width=\"100%\" Cellspacing=\"0\" Cellpadding=\"1\" Align=\"left\"><TR><TD
Bgcolor=\"${CLR_MENUBORDER}\">
<TABLE Border=\"0\" Width=\"100%\" Cellspacing=\"0\" Cellpadding=\"5\" Align=\"left\"
Valign=\"top\">
<P>
  <TR>
    <TD Bgcolor=\"${CLR_MENUBG}\" Align=\"left\" Valign=\"top\">
      <FONT Size=\"2\" Color=\"${CLR_MENUTEXT}\">
<I>Current:</I><STRONG>${PAGENAME1// / }</STRONG>
<BR><A Href=\"${PARENTID1}.shtml\"><I>Previous:</I>${PARENTNAME// / }</A>
" >> "${PAGEID1/#*\///tmp/}.menu.shtml"

####
```

```
#### If the description of the parent of the current child
#### file name is equal to the phrase "Home Page", then
#### output a link to the "index" file with a description of
#### "Home Page".
####

    if [[ "_${PARENTNAME}" != "_Home Page" ]]
    then
      print "<BR><A Href=\"index.shtml\">Home Page</A>" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
    fi
    print "<BR>" >> "${PAGEID1/#*\///tmp/}.menu.shtml"

####
#### Using the previously generated list of children of the
#### current child file name prefix, loop through each line
#### in the list and extract the three fields into
#### secondary shell
#### variables.  Again the three fields are parent file name
#### prefix, child file name prefix, and child description.
####

    while IFS=$' \t\n' read -r -- PARENTID2 PAGEID2 PAGENAME2
    do

####
#### If the child file name prefix contains a hash mark (#)
#### this means the link is a reference to an anchor tag
#### within the same file.  For this occurance, set a shell
#### variable to the exact value of the child file name
#### prefix.  Otherwise set a shell variable to the child
#### file name prefix followed by the literal characters
#### ".shtml".

      LINK="${PAGEID2}.shtml"
      if [[ "_${PAGEID2}" == _*\#* ]]
      then
        LINK="${PAGEID2}"
      fi

      (( VERBOSE == TRUE )) && print "        ${PAGENAME2}"

####
#### Output the HTML link to the menu file.
####
      print "<BR><A Href=\"${LINK}\">${PAGENAME2// / }</A>" >>
"${PAGEID1/#*\///tmp/}.menu.shtml"

      if (( GRANDCHILD == TRUE ))
      then

####
#### Re-read the parent-child relationship datafile and
#### extract the lines that begin with the secondary child
#### file name prefix and save the lines to a temporary
#### storage file for processing.  This data can be used for
#### generating grandchild links, if so specified by command
#### line option "-g".
####

        while IFS=$'\n' read -r -- DATALINE
        do
          [[ "_${DATALINE}" == _${PAGEID2}+([$'\t'])* ]] &&
            print -r -- "${PAGEID2}"
        done < "${DATATMP0}" > "${DATATMP2}"

####
#### Using the lines extracted from the parent-child
#### relationship datafile that begin with the secondary
#### child file name prefix, read each line and use the
#### information to generate grandchild HTML links that will
```

```
#### appear in the "menu" bar.  These are grandchildren of
#### the current child file name prefix.
####

        ICNT="${FALSE}"
        print "<FONT Size=\"1\">" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
        while IFS=$' \t\n' read -r -- PARENTID3 PAGEID3 PAGENAME3
        do
          ICNT="${TRUE}"
          LINK="${PAGEID3}.shtml"

####
#### If the grandchild file name prefix contains a hash (#)
#### character, then this page should link to an anchor tag
#### associated with it's parent.
####

          if [[ "_${PAGEID3}" == _*\#* ]]
          then
            LINK="${PARENTID3}.shtml${PAGEID3}"
          fi

          print "            ${PAGENAME3}"

#### Display the grandchild link in the menu bar.

          print "  <LI> <A Href=\"${LINK}\">${PAGENAME3// / }</A></LI>" >>
"${PAGEID1/#*\///tmp/}.menu.shtml"

        done < "${DATATMP2}"

        print "</FONT>" >> "${PAGEID1}.menu.shtml"
        (( ICNT == TRUE )) && print "<BR>" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
      fi
    done < "${DATATMP1}"

#################################################################
    print "
      </FONT>
    </TD>
  </TR>
</P>
</TABLE>
</TD></TR></TABLE>
</TD></TR></TABLE>
</P>
" >> "${PAGEID1/#*\///tmp/}.menu.shtml"

####
#### If the "underline" option was specified on the command
#### line, output a style tag to cause HTML links in the menu
#### bar to be turned off.
####

    if (( UNDERLINE == TRUE ))
    then
      print "
<STYLE>
<!--
        A { Color:${CLR_LINK}; text-decoration:underline; }
        A:hover { text-decoration:underline;color:${CLR_LINK}; }
//-->
</STYLE>
" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
    fi

    print "
<!-- End included menu file \"${PAGEID1}.menu.shtml\" -->
" >> "${PAGEID1/#*\///tmp/}.menu.shtml"
```

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

####
#### If the "-R" option was specified on the command line,
#### format the HTML to render the "menu" bar on the
#### right hand side of the page, otherwise render the
#### "menu" bar on the left hand side of the page.
####

    if [[ "_${MENUPOS}" = "_right" ]]
    then
      print "
<P><TR><TD Valign=\"top\">
<!--#include file=\"${PAGEID1}.content.shtml\" -->
</TD></TR></P>
</TABLE>
</P>
" >> "${PAGEID1/#*\///tmp/}.shtml"

    else

      print "
<!--#include file=\"${PAGEID1}.content.shtml\" -->
" >> "${PAGEID1/#*\///tmp/}.shtml"

    fi

    print "<!--#include file=\"footer.shtml\" -->
</BODY>
</HTML>
" >> "${PAGEID1/#*\///tmp/}.shtml"

####
#### If the "-n" option was specified on the command line,
#### then in addition to generating SSI ".shtml" pages, also
#### generate ".html" pages with all included pages inserted
#### into the final document.
####

    if (( HTMLFILES == TRUE ))
    then
      print "
<HTML>
<!-- PARENTID1=\"${PARENTID1}\" -->
<!-- PAGEID1=\"${PAGEID1}\" -->
<!-- PAGENAME1=\"${PAGENAME1}\" -->
<HEAD>
<TITLE>${PAGENAME1} - ${THEME}</TITLE>
</HEAD>
<BODY Bgcolor=\"${CLR_BACKGRD}\" Text=\"${CLR_TEXT}\" Link=\"${CLR_LINK}\" Alink=\"${CLR_ALINK}\"
Vlink=\"${CLR_VLINK}\">
<BASEFONT Size=\"${SIZ_BASEFONT}\">
" > "${PAGEID1/#*\///tmp/}.html"

####
#### Read the HTML code from the "staticbar.shtml" file and
#### insert it into the ".html" file.  Do the same for the
#### "titlebar.shtml" and "hr.shtml" files.
####

      cat_file "staticbar.shtml" >> "${PAGEID1/#*\///tmp/}.html"
      cat_file "titlebar.shtml" >> "${PAGEID1/#*\///tmp/}.html"
      cat_file "hr.shtml" >> "${PAGEID1/#*\///tmp/}.html"

####
#### If the "-R" option was specified on the command line,
#### format the HTML to render the "menu" bar on the right
#### hand side of the ".html" page, otherwise render the
#### "menu" bar on the left hand side of the page.
```

```
      if [[ "_${MENUPOS}" = "_right" ]]
      then

        print "
<TABLE Width="100%" Cellspacing=\"0\" Cellpadding=\"0\" Border=\"0\">
<P><TR><TD Width=\"80%\" Height=\"10%\">
<P><H2>${PAGENAME1}</H2></P>
<P><HR></P>
</TD><TD Width="20%" Align=\"right\" Valign=\"top\" rowspan=\"2\">
" >> "${PAGEID1}.html"

#         sed -e "s/\.shtml/.html/g" "${PAGEID1}.menu.shtml"  >> "${PAGEID1/#*\///tmp/}.html"
# replaced by following while loop

#### Read the "menu" HTML code and insert it into the ".html"
#### file.

        while IFS=$'\n' read -r -- DATALINE
        do
          print "${DATALINE//.shtml/.html}"
        done < "${PAGEID1}.menu.shtml" >> "${PAGEID1/#*\///tmp/}.html"

        print "
</TD></TR></P>
<P><TR><TD Valign=\"top\">
" >> "${PAGEID1/#*\///tmp/}.html"

#### Read the "content" file and insert it into the ".html"
#### file.

        cat_file "${PAGEID1}.content.shtml"  >> "${PAGEID1/#*\///tmp/}.html"

        print "
</TD></TR></P>
</TABLE>
</P>
" >> "${PAGEID1/#*\///tmp/}.html"

      else

####
#### If the "-R" option was NOT specified on the command line,
#### format the HTML to render the "menu" bar on the left
#### hand side of the ".html" page.

#         sed -e "s/\.shtml/.html/g" "${PAGEID1}.menu.shtml"  >> "${PAGEID1/#*\///tmp/}.html"
# replaced by following while loop

#### Read the "menu" file and insert it into the ".html"
#### file.

        while IFS=$'\n' read -r -- DATALINE
        do
          print "${DATALINE//.shtml/.html}"
        done < "${PAGEID1}.menu.shtml" >> "${PAGEID1/#*\///tmp/}.html"

#### Insert the page title into the ".html" file

        print "<P><H2>${PAGENAME1}</H2></P>
<P><HR></P>
" >> "${PAGEID1/#*\///tmp/}.html"


#### Insert the page content into the ".html" file

        cat_file "${PAGEID1}.content.shtml"  >> "${PAGEID1/#*\///tmp/}.html"
      fi

#### Insert the page divider or horizontal rule and page
#### footer  into the ".html". file
```

```
      cat_file "hr.shtml"  >> "${PAGEID1/#*\///tmp/}.html"
      cat_file "footer.shtml"  >> "${PAGEID1/#*\///tmp/}.html"

#### End the ".html" file with the appropriate HTML tags

      print "</BODY></HTML>" >> "${PAGEID1/#*\///tmp/}.html"

    fi

  done < ${DATAFILE}

#### Cleanup any remaining temporary files

  rm -f "${DATATMP0}"
  rm -f "${DATATMP1}"
  rm -f "${DATATMP2}"
}
################################################################

menugen "${@}"
```

Console Access

Privileged Access

The term “Privileged Access” refers to capabilities beyond that granted to a normal user, and in the context of “Business Continuity” this must be controlled and monitored. There are numerous reasons to provide “privileged access” to a user such as system configuration, user management, printer management, application installation and management, network management, etc. The level of access granted will depend upon who the user is, what tasks the user needs to perform, and how often the user needs to perform these tasks. AIX provides some built in mechanisms through the use of groups to provide some of this.

System Administrator

The "*System Administrator*" by default has full access to all system resources, functions, and content. The user ID used for this purpose is "*root*". Access to this login and password should be strictly reserved for System Administrators responsible for the operation and maintenance of the system. No one outside this realm of responsibility should be able to login to any AIX machine as "*root*" or have access to the "*root*" password.

Printer Management

Application administrators will need the ability to manage and enable/disable printers. This level of administration can be granted by adding the user name to the "*printq*" group. This does not provide any other system or application privileges and may be granted to those application users who are AIX literate.

User Management

The system administrator(s) for each machine and members of the your organization's "Security" group will require administrative privileges which provide user management capabilities. These privileges can be granted by adding the user to the "*security*" group. This will allow the user the ability to create, modify, and remove users from a system. They will also allow the ability to reset passwords, unlock a "locked" account, and reset a users failed login count.

From time-to-time vendors, contractors, consultants, and application administrators will need "*root*" access to one or more AIX machines. In order to provide this access, we must analyze and segment the individual requirements and merits of each request.

"sudo" Access

For those users who need to run a small set of specific commands as "*root*", they should be granted "sudo" access. "sudo" is a commonly used freeware application that provides normal users with limited advanced access privileges, and can be closely monitored and controlled. The system administrator must configure "sudo" access on each machine and assign privileges to each user to run each required command.

For users who need to run a larger set of commands as "root" or who will require privileged access for an extended period of time, other mechanisms may be better suited for this purpose. The following mechanisms are not recommended and will cause systems to fail security audits, but are described here simply for academic purposes.

UID "0"

Any user on an AIX system which has UID number "0" (zero), is regarded as the "root" user. The name of the the user with UID "0" (zero) is irrelevant, therefore the UID of any user can be changed to "0" (zero) to provide privileged access, without compromising the "root" password. To change the UID number of a user, edit the

"/etc/passwd" file and enter a "0" (zero) in the third field of the record associated with the user.

"ash" Group

Another mechanism to provide privileged access to all system commands utilizes the "set UID" permission bit. Any executable binary file with it's "set UID" bit turned on will cause the file to be executed as the owner of the file, whenever the file is executed by any user. Therefore an executable binary file with it's "set UID" bit turned on and owned by "`root`", when executed by any user, will run as though it were executed by "`root`".

Turning on the "set UID" bit on all system level commands would provide privileged access, however that would be a very dangerous and undesirable thing to do. A desirable thing to do may be to turn on the "set UID" bit of a single program, from which we can gain privileged access to system level commands. We can achieve this through the following procedure:

```
if [[ ! -f /usr/bin/ash ]]
then
     mkgroup ash
     cp /usr/bin/ksh /usr/bin/ash
     chown root:ash /usr/bin/ash
     chmod 4550 /usr/bin/ash
fi
```

This procedure creates a group called "ash", then copies the korn shell executable binary file to a file called "ash". The ownership of the file is changed to be owned by "`root`" and a member of the group "`ash`". The "set UID" permission bit for the file "/usr/bin/ash" is turned on and execute permissions are limited to "`root`" and members of the group "`ash`". No other users are permitted to execute this file. Now any user that is a member of the group "ash" can execute the file "/usr/bin/ash" to gain "root" level access privileges. Privileged user access can be controlled by adding or removing users from the "`ash`" group and preserves "`root`" password integrity.

Again, the above is NOT a recommended procedure and will likely cause more problems than it solves, but it can be and is used occasionally.

Job Scheduling

Many mechanisms for job scheduling are available for an AIX system, the most common of course is “cron”. All AIX system administrators should be familiar with “cron” and as such it will not be discussed here.

A Business Continuity mentality requires the consideration of cross platform job scheduling systems, because they provide the ability to easily redirect and reschedule jobs from one system or data center, to another system or data center. The problem with cross-platform job scheduling systems is they usually require an agent program running on each machine with “root” level privileges. This privilege level is required so the agent can execute a scheduled job as whatever user is required by the job.

By their very nature, a cross-platform scheduling system is a security risk, it may be a necessary risk for your organization, but business management and IT management need to be aware of the risk, and understand the implications to your organization.

An real world example of this danger with cross-platform scheduling systems (a specific product will not be named, but it was NOT an IBM product) follows:

An international organization utilizing multiple platforms in multiple data centers required a scheduling system to facilitate data flow between the various systems. These systems included:

- Multiple IBM mainframes
- 4 AS/400's
- 200 AIX systems
- 100 MS Windows Servers

To implement the job scheduling system required an agent be installed on each system, which was done. Once installed the system work beautifully, when a job on one system was complete, messages were being passed to the next system responsible for processing the data, and it took over, etc., etc.

Unfortunately it was observed that any system connected to the network with this job scheduling agent installed had the capability of scheduling any job on any system in the organization, to run as any user. Which meant that a Windows administrator could schedule a job on the mainframe, AS/400, or AIX system as any user they desired, without having user access to that system, and vice versa.

Furthermore, the only thing necessary to schedule a job on any system as any user, was to have the job scheduling agent installed on any desktop system, and the agent was easily obtainable. Which meant that anyone with this easily obtainable agent could connect to the network and have access to any system, application or data. If your organization uses an enterprise scheduler, access and security to the participating systems must be closely controlled and monitored.

AIX CRON

The "cron" daemon is a tried and true tool for scheduling on all Unix systems and is utilized in nearly all organizations that implement Unix systems. The "cron" daemon is configured using a standard text files called a "crontab", which contains scheduling and command execution information. The "crontab" file provides the system administrator the ability to schedule a job to run as any user, at any time of day, on any day. However, the AIX "crontab" does not provide the ability to execute a job, for example, on the second Sunday of the month. If you attempted to schedule this in an AIX "crontab", the record might appear as:

```
0    0    8,9,10,11,12,13,14          *    0    command_to_run
```

The above "crontab" record would NOT execute the job on the second sunday of the month because the AIX "crond" assumes an "OR" between the monthday and the weekday fields of each record. The result of the above record would be the job "command_to_run" would be executed on the 8th, 9th, 10th, 11th, 12th, 13th, and 14th of every month OR on every Sunday. To be able to schedule a job to be executed on the second Sunday of each month requires the "crond" daemon to assume an "AND" between each field of the record. The result of assuming an "AND" between each field of the above "crontab" record would be the job would be executed only on Sunday when the day of the month is the 8th, 9th, 10th, 11th, 12th, 13th, or 14th , which equates to the second Sunday of each month.

Specifying any single weekday and a monthday of 1-7 is equivalent to the first occurrence of that weekday in the month. The second occurrence of any single weekday will always occur on a monthday between 8-14, the third occurrence of any single weekday will always occur on a monthday between 15-21, and the fourth occurrence on a monthday between 22-28.

Unfortunately, the AIX cron daemon does not permit the ability to specify the "AND" conjunction between the weekday and monthday fields of a crontab record. However, the following "crond" emulation script does:

```
#!/usr/bin/ksh93
#################################################################
function usagemsg_ecrond_k93 {
  print "
Program: ecrond_k93

Usage: ${1##*/} [-?] [-vV] [-a] [-o] [-d crontabDir]
                    [-f crontabFile]

    Where:
                -a = Use "AND" between day of month and day
                     of week to determine match.
                -o = Use "OR" between day of month and day
                     of week to determine match. (Default)
    -d crontabDir  = Directory where crontabs are stored.
                     Default: /usr/spool/ecron_k93/crontabs
    -f crontabFile = Name of a file containing crontab records.
                     If specified, only this file will be processed.
                     Default: none
                -v = Verbose mode
                -V = Very Verbose Mode

Example: ecrond_k93 -v -a -d "${HOME}/crontab"

Author: Dana French (dfrench@mtxia.com)
        Copyright 2006 by Dana French

\"AutoContent\" enabled
"
}
#################################################################
####
#### Description:
####
#### Place a full text description of your shell function here.
####
#### Assumptions:
####
#### Provide a list of assumptions your shell function makes,
#### with a description of each assumption.
####
#### Dependencies:
####
#### Provide a list of dependencies your shell function has,
#### with a description of each dependency.
####
#### Products:
####
#### Provide a list of output your shell function produces,
#### with a description of each product.
####
#### Configured Usage:
####
#### Describe how your shell function should be used.
####
#### Details:
####
#### Place nothing here, the details are your shell function.
####
#################################################################
function mkdigarry
{
  nameref FIELD="${1}"
  nameref DIGARY="${2}"
  typeset RNG  DIG  BEG  END

  IFS=",:"
  RNGARY=( ${FIELD} )
```

```
  for RNG in "${RNGARY[@]}"
  do
   if [[ "_${RNG}" == _+(?)$'-'+(?) ]]
   then
      BEG="${RNG%%-*}"
      END="${RNG##*-}"
      (( BEG > END )) &&
          print -u 2 -- "# ERROR: ${0}: Invalid range: ${RNG}" &&
          return 1
      for (( DIG=${BEG}; DIG<=${END}; ++DIG ))
      do
          DIGARY[${DIG}]="true"
      done
   else
      DIGARY[${RNG}]="true"
   fi
  done
  return 0
}
################################################################
function ecrond_k93 {
  typeset TRUE="1"
  typeset FALSE="0"
  typeset RETCODE="0"
  typeset VERBOSE="${FALSE}"
  typeset VERYVERB="${FALSE}"
  typeset VERSION="1.0"
  typeset DOMDOW="${FALSE}"
  typeset CRONFILE="${FALSE}"
  typeset DATESTAMP=$( date +"%Y%m%d%H%M%w" )
  typeset CURMON="${DATESTAMP:4:2}"
  typeset CURDOM="${DATESTAMP:6:2}"
  typeset CURHOUR="${DATESTAMP:8:2}"
  typeset CURMIN="${DATESTAMP:10:2}"
  typeset CURDOW="${DATESTAMP:12}"
  typeset DD_CRONTAB="/usr/spool/ecron_k93/crontabs"
  typeset -Z1 DOW
  typeset -Z2 MON  DOM  HOUR  MIN
  typeset F1  F2  F3  F4  F5
  typeset CMDLINE
  typeset CRONTAB
  typeset CRONLOG="/tmp/ecrond_k93.log"
  typeset CRONTABFILE=""
  typeset UID="$( id_k93 -u )"

  while getopts ":vVoad:f:" OPTION
  do
    case "${OPTION}" in
      'o') DOMDOW="${FALSE}";;
      'a') DOMDOW="${TRUE}";;
      'd') DD_CRONTAB="${OPTARG}";;
      'd') DD_CRONTAB="${OPTARG}";;
      'f') CRONFILE="${TRUE}"
           CRONTABFILE="${OPTARG}";;
      'v') VERBOSE="${TRUE}";;
      'V') VERYVERB="${TRUE}";;
      '?') usagemsg_ecrond_k93 "${0}" && return 1 ;;
      ':') usagemsg_ecrond_k93 "${0}" && return 1 ;;
      '#') usagemsg_ecrond_k93 "${0}" && return 1 ;;
    esac
  done

  shift $(( ${OPTIND} - 1 ))

  (( VERBOSE == TRUE )) && print -u 2 -- "# Version: ${VERSION}"

################################################################

  trap "usagemsg_ecrond_k93 ${0}" EXIT
```

```
  RETCODE="1"
  if (( CRONFILE == FALSE )) && [[ ! -d "${DD_CRONTAB}" ]]
  then
      print -u 2 -- "# ERROR: Specfied crontab directory is invalid: ${DD_CRONTAB}"
      return ${RETCODE}
  fi

  RETCODE="2"
  if (( CRONFILE == TRUE )) && [[ ! -f "${CRONTABFILE}" ]]
  then
      print -u 2 -- "# ERROR: Specfied crontab file is invalid: ${CRONTABFILE}"
      return ${RETCODE}
  fi

  RETCODE="0"

  trap "-" EXIT

  (( VERYVERB == TRUE )) && set -x

################################################################

#   print -r -- "curmin=${CURMIN}"
#   print -r -- "curhour=${CURHOUR}"
#   print -r -- "curdom=${CURDOM}"
#   print -r -- "curmon=${CURMON}"
#   print -r -- "curdow=${CURDOW}"

  (( CRONFILE == FALSE )) && FILELIST=( ${DD_CRONTAB}/* )
  (( CRONFILE == TRUE  )) && FILELIST=( ${CRONTABFILE} )

  for CRONTAB in "${FILELIST[@]}"
  do
    if [[ "_$( id_k93 -un ${CRONTAB##*/} )" == "_" ]]
    then
        print -u 2 -- "ERROR: Crontab file is not named for a valid user."
        continue
    fi
    if (( UID != 0 ))
    then
        if [[ ! -O "${CRONTAB}" ]]
        then
            print -u 2 -- "# ${CRONTAB} is not owned by the current user"
            continue
        fi
    fi

    [[ ! -s "${CRONTAB}" ]] && continue

    (( VERBOSE == TRUE)) && print -r -u 2 -- "# Crontab file: ${CRONTAB}"

    while IFS=$' \t\n' read -r -- F1 F2 F3 F4 F5 CMDLINE
    do
      [[ "_${F1}" == '_*' ]] && F1="0-59"
      [[ "_${F2}" == '_*' ]] && F2="0-23"
      [[ "_${F3}" == '_*' ]] && F3="1-31"
      [[ "_${F4}" == '_*' ]] && F4="1-12"
      [[ "_${F5}" == '_*' ]] && F5="0-6"

      unset  MINS  HRS  MDAYS  MONS  WDAYS
      set -A MINS
      set -A HRS
      set -A MDAYS
      set -A MONS
      set -A WDAYS

      mkdigarry F1 MINS || continue
      mkdigarry F2 HRS || continue
      mkdigarry F3 MDAYS || continue
```

```
mkdigarry F4 MONS || continue
mkdigarry F5 WDAYS || continue

(( MATCH = FALSE ))

for MON in "${!MONS[@]}"
do
    if (( ( MON >=  1 ) &&
          ( MON <= 12 ) &&
          ( MON == CURMON ) ))
    then
        for HOUR in "${!HRS[@]}"
        do
            if (( ( HOUR >=  0 ) &&
                  ( HOUR <= 23 ) &&
                  ( HOUR == CURHOUR ) ))
            then
                for MIN in "${!MINS[@]}"
                do
                    if (( ( MIN >=  0 ) &&
                          ( MIN <= 59 ) &&
                          ( MIN == CURMIN ) ))
                    then
                        if (( DOMDOW == TRUE ))
                        then
                            for DOW in "${!WDAYS[@]}"
                            do
                                if (( ( DOW >=  0 ) &&
                                      ( DOW <=  6 ) &&
                                      ( DOW == CURDOW ) ))
                                then
                                    for DOM in "${!MDAYS[@]}"
                                    do
                                        if (( ( DOM >=  1 ) &&
                                              ( DOM <= 31 ) &&
                                              ( DOM == CURDOM ) ))
                                        then
                                            (( MATCH = TRUE ))
                                            break 5
                                        fi    # DOM
                                    done    # DOM
                                fi    # DOW
                            done    # DOW
                        else
                            for DOW in "${!WDAYS[@]}"
                            do
                                if (( ( DOW >=  0 ) &&
                                      ( DOW <=  6 ) &&
                                      ( DOW == CURDOW ) ))
                                then
                                    (( MATCH = TRUE ))
                                    break 4
                                fi    # DOW
                            done    # DOW
                            for DOM in "${!MDAYS[@]}"
                            do
                                if (( ( DOM >=  1 ) &&
                                      ( DOM <= 31 ) &&
                                      ( DOM == CURDOM ) ))
                                then
                                    (( MATCH = TRUE ))
                                    break 4
                                fi    # DOM
                            done    # DOM
                        fi    # DOMDOW
                    fi    # MIN
                done    # MIN
            fi    # HOUR
        done    # HOUR
    fi    # MON
```

```
      done    # MON

      if (( MATCH == TRUE ))
      then
        print -r -- "MATCH=TRUE"
        print -r -- "${MIN} ${HOUR} ${DOM} ${MON} ${DOW} ${CMDLINE}"

        if (( UID == 0 ))
        then
            print -- su - ${CRONTAB##*/} -c "nohup ${CMDLINE} >> ${CRONLOG}"
            su - ${CRONTAB##*/} -c "nohup ${CMDLINE} >> ${CRONLOG} 2>&1 &"
        else
            print -- nohup ksh93 -- "${CMDLINE} >> ${CRONLOG} 2>&1 &"
            nohup ksh93 -- "${CMDLINE}" >> ${CRONLOG} 2>&1 &
        fi
      fi

    done < "${CRONTAB}"
  done

  return ${RETCODE}
}
#################################################################
####
#### Description:
####
#### The id_k93 function writes to standard output a message
#### containing the system identifications (ID) for a
#### specified user.  The system IDs are numbers which
#### identify users and user groups to the system.
####
#### Details:
####
#################################################################
function id_k93 {
  typeset TRUE="0"
  typeset FALSE="1"
  typeset VERBOSE="${FALSE}"
  typeset VERYVERB="${FALSE}"
  typeset ONLYGID="${FALSE}"
  typeset ONLYGRP="${FALSE}"
  typeset ONLYUSR="${FALSE}"
  typeset REALIDS="${FALSE}"
  typeset NAMEOUT="${FALSE}"
  typeset USERNAME
  typeset PWDUSER
  typeset PWDPWD
  typeset PWDUID
  typeset PWDGID
  typeset PWDGECOS
  typeset PWDHOME
  typeset PWDSHELL
  typeset EUID
  typeset EGID

  while getopts ":GgurnvV" OPTION
  do
    case "${OPTION}" in
        'G') ONLYGID="${TRUE}";;
        'g') ONLYGRP="${TRUE}";;
        'u') ONLYUSR="${TRUE}";;
        'r') REALIDS="${TRUE}";;
        'n') NAMEOUT="${TRUE}";;
        'v') VERBOSE="${TRUE}";;
        'V') VERYVERB="${TRUE}";;
        '?') usagemsg_id_k93 "${0}" && return 1 ;;
    esac
  done

  shift $(( ${OPTIND} - 1 ))
```

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

####
#### Obtain the target username from the first command line
#### argument, if NULL determine the target username from
#### the existing environment variable LOGIN.  If this value
#### is NULL use LOGNAME, if still NULL use USER.  If still
#### NULL exit with an error message.
####

  USERNAME="${1:-${LOGIN}}"
  USERNAME="${USERNAME:-${LOGNAME}}"
  USERNAME="${USERNAME:-${USER}}"
  USERNAME="${USERNAME:?ERROR: Unable to determine user name}"

####
#### Read each line of the /etc/passwd file and match the
#### target username against each username listed in the
#### file.  When a match is found, save the UID and GID
#### values for later use and stop processing the file.
####

  while IFS=":" read -- PWDUSER PWDPWD PWDUID PWDGID PWDGECOS PWDHOME PWDSHELL
  do
    if [[ "_${PWDUSER}" = "_${USERNAME}" ]]
    then
        EUID="${PWDUID}"
        EGID="${PWDGID}"
        break
    fi
  done < /etc/passwd

  if [[ "_${EUID}" == "_" ]] || [[ "_${EGID}" == "_" ]]
  then
      print -u 2 -- "# ${0}: ${USERNAME}: No such user"
      return 1
  fi

#### If the "-u" option was entered on the command line,
#### output the UID number and exit the function.

  (( ONLYUSR == TRUE && NAMEOUT == FALSE )) && print "${EUID}" && return 0

#### If the "-u" and "-n" options were entered on the
#### command line, output the username and exit the function.

  (( ONLYUSR == TRUE && NAMEOUT == TRUE )) && print "${USERNAME}" && return 0

#### If the "-g" option was entered on the command line,
#### output the GID number and exit the function.

  (( ONLYGRP == TRUE && NAMEOUT == FALSE )) && print "${EGID}" && return 0

####
#### Read each line from the /etc/group file to determine the
#### group name associated with the GID number extracted from
#### the user record in the /etc/passwd file.  Assign this
#### group name as the primary group.
####

  while IFS=":" read -- GRPNAME GRPPWD GRPGID GRPUSERS
  do
    if [[ "_${EGID}" = "_${GRPGID}" ]]
    then
        PRIMARYGROUP="${GRPNAME}"
        break
    fi
  done < /etc/group
```

```
#### If the "-g" and "-n" options were entered on the
#### command line, output the primary group name and exit the function.

  (( ONLYGRP == TRUE && NAMEOUT == TRUE )) && print "${PRIMARYGROUP}" && return 0

####
#### To ensure empty arrays, incase multiple user names are
#### allowed on the command line, unset the arrays used to
#### contain the list of secondary group names and GID
#### numbers.
####

  unset GRPNAMLIST
  unset GRPGIDLIST

####
#### Use a counter to keep track of the GID order in the
#### array.  Since the GID number may be greater than the
#### allowed size of a korn shell array, a separate counter
#### is used.
####

  CNT=0

####
#### Loop through each line of the /etc/group file and
#### determine what groups the username is associated with.
#### When a group is found, store the group name and the GID
#### number in separate arrays to preserve the numerical
#### order.
####

  while IFS=":" read -- GRPNAME GRPPWD GRPGID GRPUSERS
  do
    (( GRPGID == EGID )) && continue
    if [[ "_${GRPUSERS}" = _${USERNAME} ]] ||
       [[ "_${GRPUSERS}" = _${USERNAME},* ]] ||
       [[ "_${GRPUSERS}" = _*,${USERNAME},* ]] ||
       [[ "_${GRPUSERS}" = _*,${USERNAME} ]]
    then
        GRPNAMLIST[CNT]="${GRPNAME}"
        GRPGIDLIST[CNT]="${GRPGID}"
        (( CNT++ ))
    fi
  done < /etc/group

#### If the '-G' option was entered on the command line,
#### print the GID number of the primary group.  If the '-n'
#### option was also used, print the primary group name,
#### otherwise print the normal 'id' command output.

  if (( ONLYGID == TRUE && NAMEOUT == FALSE ))
  then
      print -r -n -- "${EGID}"
  elif (( ONLYGID == TRUE && NAMEOUT == TRUE ))
  then
      print -r -n -- "${PRIMARYGROUP}"
  else
      print -r -n -- "uid=${EUID}(${USERNAME}) gid=${EGID}(${PRIMARYGROUP})"
      (( ${#GRPGIDLIST[*]} != 0 )) && print -r -n -- " groups="
  fi

####
#### Loop through each stored secondary GID associated with the
#### user and print as specified by the command line options.
####

  COMMA=""
  for GRP in "${!GRPGIDLIST[@]}"
  do
```

```
####
#### If the '-G' option was specified on the command line,
#### the print the secondary group GID number.  If the '-n'
#### option was also specified, print the secondary group
#### name.  Otherwise print the normal 'id' command output.
####

    if (( ONLYGID == TRUE && NAMEOUT == FALSE ))
    then
      print -r -n -- " ${GRPGIDLIST[${GRP}]}"
    elif (( ONLYGID == TRUE && NAMEOUT == TRUE ))
    then
      print -r -n -- " ${GRPNAMLIST[${GRP}]}"
    else
      print -r -n "${COMMA}${GRPGIDLIST[${GRP}]}(${GRPNAMLIST[${GRP}]})"
      COMMA=","
    fi
  done

#### Up to this point, no CR characters have been printed,
#### since all output is now complete, send a CR character.

  print

#### Return to the calling function with a successful
#### return code.

  return 0
}
################################################################

ecrond_k93 "${@}"
```

The “ecrond_k93” crond emulation script is designed to be executed as a job scheduled to run every minute from the regular AIX crontab. It can be scheduled in the “root” crontab, however it is recommended that, if used, it be scheduled from a regular user account “crontab”. Also it should be used on a limited basis because it will be run every minute of every hour of every day.

Business Continuity Mentality

One of the main points that should be taken from this series of articles, is that Business Continuity is not a project that is performed, documented, and then stored away on a bookshelf where it is kept in case it should become necessary to use someday. Business continuity is a way of conducting business on a daily basis. It is a mentality that must be adopted by all personnel in an organization. And in order for that to happen, it must be a priority for management, if management is not committed to making it happen, don't start the process. Without management involvement and commitment, it is a waste of time, energy, and resources to attempt to implement this methodology.

Sometimes it is difficult to gauge the level of management's commitment to adopting a mentality of business continuity, and this is usually because they do not understand

that it is an enterprise wide change in the way business is conducted. When discussing the adoption of a business continuity methodology with management, stress the point that it is an enterprise wide methodology and will require the participation of all personnel in the organization. Explain that this is a business methodology, a change in the way all business is conducted, not a project.

Project Planning

An enterprise wide mentality of business continuity means that all future projects will be designed, planned, implemented, and supported with this concept in mind. All project plans will have business continuity (BC) and disaster recovery (DR) components. The specific BC/DR tasks included in all project plans will be dependent upon the methodologies adopted by your organization, but will probably contain many of the following:

- Business Impact Analysis
- Determine criticality of this project to BAU (1,2,3)
- Identify dependencies
- Identify prerequisites
- Determine Recovery Time Objective (RTO)
- Determine Recovery Point Objective (RPO)
- Determine Decision Lead Time (DLT)
- Determine DR provisioning requirements
    - Hardware
    - Software
    - Facilities
    - Backup and Restore
- Provisioning for DR
    - High Availability
        - Identify and eliminate single points of failure (SPOF's)
    - Disaster Recovery
        - Allocate funds and resources for lifetime of business function being implemented.
- Maintenance and support
    - Allocate resources for integration into existing support systems

Conclusion:

Business continuity consists of those activities that provide order and sense to the IT architecture. Those activities include the adoption of policies, guidelines, standards and procedures for designing, implementing, managing and supporting your environment. These result of these activities should be the basis of all work performed by the IT department, furthermore it is the duty of the IT department to ensure these rules are followed enterprise wide.

Some of the policies and standards identified in this series of articles:

Policies:

- Each entity defined in this environment will be configured with an enterprise wide unique identifier so that it may be moved or reconfigured anywhere in the environment.

- The Network Information Manager (NIM) is utilized for providing access to all operating system components.

- The configuration and implementation of all components comprising each LPAR is documented with both hard and soft copies of the documentation.

Standards:
The purpose of these standards is to ensure business continuity during normal system maintenance, planned and unplanned outages, hardware and software failures, network and communication failures, and/or a disaster recovery implementation.

A design aspect of these standards is they can be implemented in a standalone, high availability, or disaster recovery scenario. Recognize that there are not multiple standards, one for each scenario, there is one single standard that is portable across all scenario's. This reduces support and training costs, and increases efficiency, supportability, recoverability, and availability.

Some of the basic concepts of these standards:

- Business functions are not tied to a specific machine.
- Hardware resources can be shared or distributed among associated business functions.
- Any system can act as a failover for any other system.
- Any data center can act as a disaster recovery site for any other data center.

The result of this series of articles on the topic of "Successful Business Continuity" should be the recognition that business continuity is not a project to be staffed and performed, and then forgotten. Business continuity is way of doing business on a regular basis, it is business-as-usual. In order to ensure business continuity, organizations must design, implement, maintain and enforce policies, procedures, standards, and guidelines that encompass all aspects of their critical business functions. Business continuity must be the beginning point in systems design, not the ending point. Unfortunately, very few systems are built starting from the business continuity perspective and working backwards.

