APPMAP Application Roadmap

By – Bob Carozzoni and Jim Riviello, Systems Engineers, SMCC

APPMAP is a tool which allows system administrators to build an ASCII table to control application startup. The goal of APPMAP is to simplify and centralize administration of applications in any UNIX environment, with a particular focus on environments which are migrating from Solaris 1.*x* to Solaris 2.*x*. For less experienced administrators it provides a simple alternative to shell scripts, and for more experienced users and administrators it can augment more sophisticated approaches.

APPMAP can make the complexity of starting different applications transparent to the end user, even when there are different application revisions for different systems running different operating systems, all requiring different paths, environment variables, and startup procedures. System administrators can simply edit a text file to control and change how applications are started. Flexibility for power users to customize their environment is maintained.

The rules for how to start a particular application are controllable down to the individual user, the individual desktop, and/or the software revision desired.

The following diagram (Figure 1) provides a high-level overview of the functionality of APPMAP:

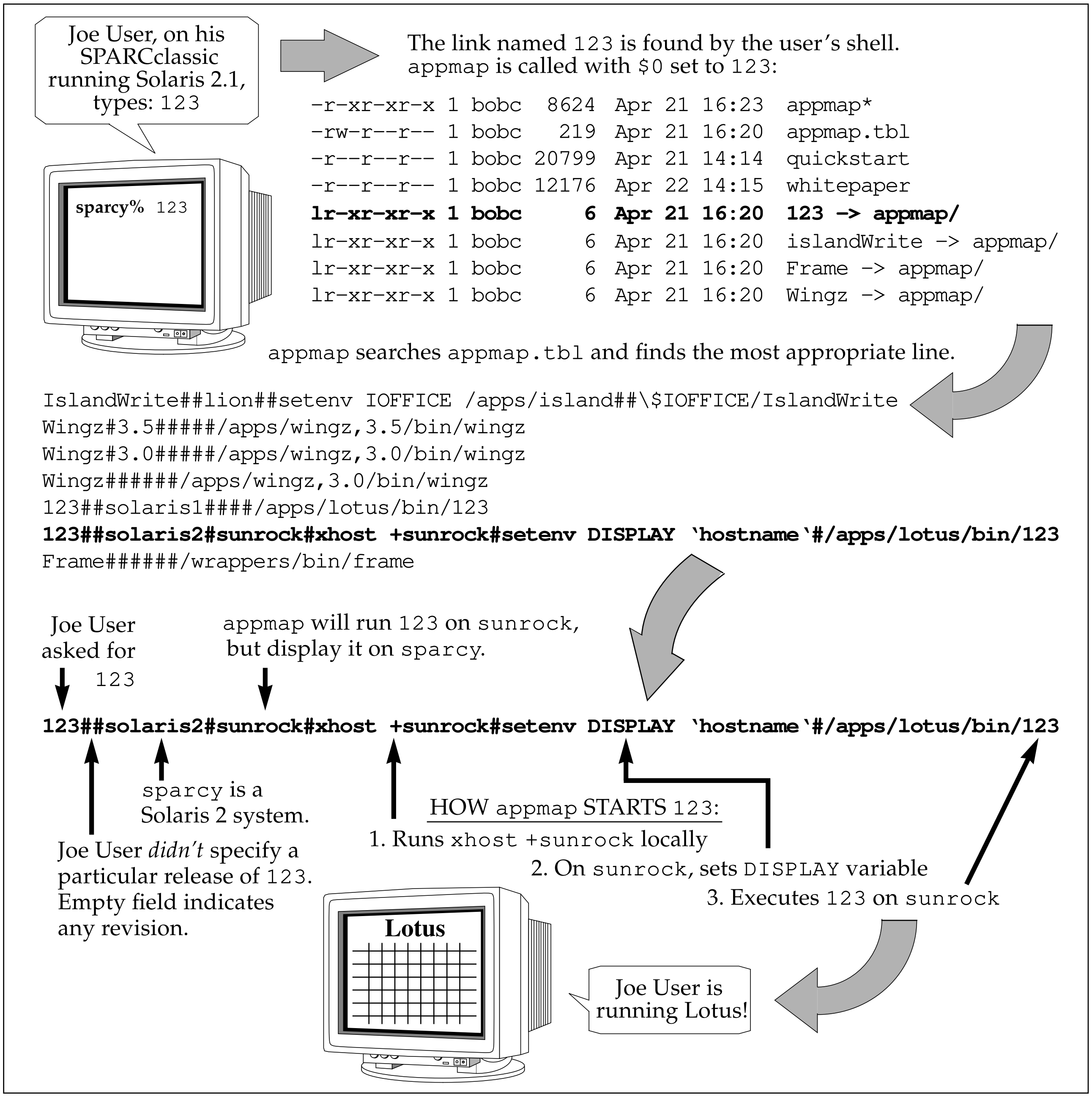

Figure 1 Overview of APPMAP

Primary Goal: Simplifying the Solaris Migration

Sun users are entering a year of transition. Their network will run a mixture of Solaris 1.*x* and 2.*x* systems. At first, some applications will run native on their Solaris 2.*x* systems, others will run only in BCP (Binary Compatibility Package) mode, and others still (if X11-based) will be run on remote Solaris 1.*x* systems for local display. Moving from one desktop to another may require users to know which OS they're running, which version of an application to start, or from which server to run the application.

Throughout the year, the status of applications and systems will change rapidly. Native ports will become available, new revisions will be released, and 1.*x* desktops will get upgraded to 2.*x* desktops. In many sites, administrators will have to continually re-educate users in proper application startup. Others who employ scripts for application management will find their scripts growing more and more complex, and it may be difficult to update their scripts fast enough.

This constant stream of changes for administrators and end users could create a great loss of productivity. APPMAP makes these changes transparent to the average users, and easier to manage for the system administrator.

Secondary Goals

- *Multiple Application Revisions*

  Many sites today maintain multiple releases of the same application, to accommodate the needs of different users. The Solaris migration makes this scenario even more likely. APPMAP simplifies the management of multiple application revisions.

- *Startup Files and Environment Variables*

  `.cshrc` and `.login` files have gotten quite complicated for many users, as nearly every application requires modification to a user's path and environment variables. When a system administrator loads a new application, he must either edit each user's startup files or notify users to do the edit themselves.

  Some administrators build system-wide `.login` or `.cshrc` files which users can `source` from their own startup files. This helps simplify a user's startup files, but may not provide for customization at the user and desktop level.

  APPMAP can be used to simplify startup files and their management, while maintaining the freedom to easily customize one's environment.

- *Loosely Managed Networks*

  Sites which have evolved without strong and centralized administrative policies often have convoluted NFS mounting maps. The full path to the same directories can be very different from system to system. This requires either (a) lots of soft links or (b) different application startup procedures for the same application, depending on the system.

  One common solution — "`if`" statements in `.login` or `.cshrc` — have made for some very complex startup files.

  APPMAP makes it easier to manage application startup in such sites.

- *Load Balancing*

  Many sites run applications on application servers and display them on the user's desktop system or X-terminal. Some applications will run only in this mode during the Solaris migration. System administrators, rather than end users, should choose which server should be used, so that they can balance the computing load among application servers.

  APPMAP gives system administrators some basic control over which servers are used, depending on (a) the application and (b) the display station. This mechanism is transparent to end users.

- *Disk Utilization*

  In order to simplify administration and to avoid complex paths, many administrators don't fully utilize the small amounts of space on desktop workstations. Unfortunately, these small pockets of unused disk space can add up to a great deal on a network with many desktops.

  Sophisticated use of the automounter is the preferred way to reclaim this disk space in a very transparent way. Where this is not an option, APPMAP can help users start applications without having to know which desktop the application is loaded on.

- *Novice System Administrators*

  Some customer sites are managed by novice system administrators, for whom complicated shell scripts (wrappers) are not an option. APPMAP is simple enough to be used in such sites.

- *Sophisticated Users*

  Automating application startup is crucial for many users, but an even greater need for sophisticated users is flexibility.

  Power users can't use tools that simplify their environment at the cost of control and flexibility. APPMAP simplifies application management for such users in a very flexible way, yet leverages existing tools where they fit best.

Strategy

Our strategy for creating APPMAP was to create a simple file format which system administrators can use to map out how applications are to be started on each system. We called the file `appmap.tbl`. Each line in `appmap.tbl` applies to a system or group of systems, and contains all information required for starting applications properly. It is possible to create a line in `appmap.tbl` which acts as a default, to reduce the number of lines needed for any particular application.

We then created a "wrapper" (shell script) that knows how to interpret `appmap.tbl` to (a) determine which line in the file is appropriate for the application, revision, and system; and then (b) start the application as indicated in the line chosen.

The wrapper looks first for a personal `appmap.tbl` file in `$HOME/appmap`, and if not found, looks for a system default `appmap.tbl` in the directory specified by the environment variable `APPMAP_HOME`.

In order to remain flexible enough to work with other existing strategies and to be easily maintained and customized, the wrapper merely automates what a user can do manually. It is still the administrator's responsibility to ensure that permissions to run an application remotely are configured properly. This design strategy was crucial to ensuring that APPMAP does nothing which affects security on the network.

Other shell scripts work if called by APPMAP.

We designed APPMAP to be easily portable, so it can be used in heterogeneous environments.

Implementation Details

The File: appmap.tbl

Entries in appmap.tbl are one line of the form:

```
app#rev#host#apphost#local_setup#remote_setup#exe
```

The field delimiter is "#" (a colon ":" was our first choice, but it conflicted with common entries in the local_setup and remote_setup fields).

The first three fields identify the application, revision, and system that the particular line pertains to. The following four fields identify how to start the application on that particular system(s).

The following describes the meaning of each field and how the appmap script interprets it:

app:
: the name of the application to be started.

rev:
: indicates the revision of the application that the user wishes to run. If the user passed a "--r ***n***" (yes, that's a double-dash) on the command line, then "***n***" is the specific revision the user wishes to run.

Note – If "--r ***n***" is not in the command line, then appmap will use the first line in appmap.tbl which contains no rev entry, and which contains a suitable app and host entry.

---

host:
: identifies an appmap.tbl entry that is specific to a particular host. Valid hostnames are acceptable entries, as well as:

  "-hostname" entries exclude a host and return an error (i.e., -saranac indicates the app is refused execution on the machine saranac).

  "solaris1" indicates any host which is running any Solaris 1.*x* release.

  "solaris2" indicates any host which is running any Solaris 2.*x* release.

  An empty host field indicates that any host is acceptable.

apphost:
: identifies the host on which the application should be run. This is used primarily for running an application across the network and displaying it locally. By associating specific servers with certain applications or desktops, a system administrator can balance the load on application servers.

local_setup:
: command(s) which are run on the local system before starting this application. Example: `xhost +`***myhost***

remote_setup:
: command(s) which are run on the `apphost` before starting this application (used only if this application is to be run remotely and displayed locally).
Example: `setenv IOFFICE /export/apps/ioffice`

exe:
: a string of commands to be executed on whichever system will run the application.

See the section below, "Example of `appmap.tbl`", for an example of an `appmap.tbl` file and guidelines for building successful `appmap.tbl` entries.

The Wrapper: appmap

"`appmap`" is implemented as a `nawk` script. It is called via a soft link, thus setting `$0` (the program-name argument) to the name of the link. The name of the link corresponds to the name of an application which is listed in the first field of the appropriate line in `appmap.tbl`.

`appmap` looks first for an `appmap.tbl` in the user's home directory (`$HOME/appmap/appmap.tbl`). This would be a personal `appmap.tbl`. If `$HOME/appmap/appmap.tbl` is not found, or if it is found but there is not a suitable entry in the file, then `appmap` searches the system default `appmap.tbl` file (`$APPMAP_HOME/appmap.tbl`).

`appmap` parses `appmap.tbl` to determine (from fields 1-3) which line in `appmap.tbl` is appropriate for the application desired. The first criteria is that the line should begin with the application name called for on the command line. The revision (if specified on the command line by `--r` ***n***) and the local system name are the two criteria for choosing between multiple lines for the same application. The first line which contains both an acceptable `rev` and `host` field is chosen, regardless of the acceptability of later lines.

`appmap` then tries to start the application according to the last four fields of the chosen line. If the `apphost` field is empty, or contains the name of the local host, the application is run locally. Otherwise, the application is run remotely.

If `--t` is specified on the command line, `appmap` will print information for testing purposes. It will report information specifying how it would have functioned if not in test mode, but nothing will actually be started.

The `--t` and `--r` options can be combined as `--rt` or `--tr`.

Command line arguments which are used by the application you wish to start (such as `cmdtool -scale large`) still function properly. The only restriction is that the `appmap`-specific arguments (`--r`, `--t`, `--rt`, or `--tr`) must appear first, before any arguments to the application. Double-dashes are used to distinguish between arguments to `appmap` and arguments for the application.

See the final section of this article for the source code of `appmap`.

Links

`appmap` is not to be called directly. For each application which is defined in `appmap.tbl`, a link should be created. The name of the link is that of the application, and the file it points to is `appmap`.

Example:

```
prompt% ln -s appmap frame
lr-xr-xr-x 1 bobc 6 Apr 21 16:20 frame -> appmap/
```

When the user types "`frame`", `appmap` will be called, with `$0` set to `frame`.

Environment Setup

The environment variable `APPMAP_HOME` should be set to the directory where `appmap` is installed. We suggest that `APPMAP_HOME` be the first entry in `PATH`. Any command which is defined in either `$HOME/appmap/appmap.tbl` or `$APPMAP_HOME/appmap.tbl` must have a corresponding link in `$APPMAP_HOME`.

If you do not wish to have APPMAP manage a particular application, you must not create a link for it. If you have link for an application which APPMAP doesn't know how to manage, `appmap` will return an error and the rest of the user's `PATH` will not be searched.

If `APPMAP_HOME` is not the first entry in `PATH`, and you wish APPMAP to manage an application, you must ensure that the application's name is not found in `PATH` before the link of the same name in `APPMAP_HOME`.

It is a matter of administrative policy if other variables, such as `OPENWINHOME`, are to be set in the `.cshrc` or `.login` files, or if the `appmap.tbl`'s `local_setup` and `remote_setup` fields contain this information. Using APPMAP to control variables like `OPENWINHOME` and `LD_LIBRARY_PATH` has the advantage of simplifying startup files. Unfortunately, doing so can make lines in the `appmap.tbl` file very long and hard to read. Another alternative is to create shell scripts which are called for from the `local_setup` and the `remote_setup` fields.

Personal `appmap`

If you maintain your own personal `appmap.tbl` in `$HOME/appmap/appmap.tbl`, you should verify that the proper links exist in `APPMAP_HOME`. If you are managing an application which is not managed by the system-wide `appmap.tbl` (and no link exists), you can do the following:

a. Create a link in `$HOME/appmap`

```
ln -s $APPMAP_HOME/appmap name
```

where ***name*** is the name of the application.

b. Then place `$HOME/appmap` first in your search path, before `$APPMAP_HOME` .

Security

Because APPMAP simply automates what users would type manually when starting an application, there are no security considerations of note. That is not to say that APPMAP could not be used by a clever hacker to compromise security, but rather that APPMAP does not create any *new* security risks. Both the risks and the countermeasures associated with APPMAP are the same as for a user's startup files.

Summary

The original motivation for APPMAP was a reaction to the complexities of running a mixed network of Solaris 1.*x* and Solaris 2.*x* systems. It soon became clear that the problems we were addressing are not unique to the Solaris migration, and in fact have always faced users and system administrators to one degree or another.

Application startup procedures can become extremely complex in sites which must manage multiple OS releases, multiple application revisions, unique user requirements and uniquely installed desktops and servers. Particularly during the Solaris migration, users and system administrators need help with the task of managing application startup. APPMAP allows users and system administrators to reduce the many permutations of application startup down to a set of rules defined in an easily maintained ASCII file. By simplifying the management of applications, APPMAP allows users and system administrators to spend their time more productively.

One of the greatest challenges in designing APPMAP was resisting the temptation to try to do too much. APPMAP is, by design, a very simple solution that relies heavily on the administrator who uses it. From that simplicity comes the flexibility and versatility to be used in many varied situations.

We, the authors, have found APPMAP to be quite useful on our own networks. We feel that the utility of APPMAP for simplifying the administration of applications is limited primarily by our ingenuity in building `appmap.tbl` files. We're continually finding new and interesting ways to apply APPMAP in our environment.

Note – APPMAP is being submitted to SunSITE. If you can access the Internet, you can `ftp` to `SunSITE.unc.edu` using the account "`anonymous`" to download a copy of APPMAP. The package will contain this paper, the `appmap` script, a sample `appmap.tbl` file, a quickstart guide, and a list of known bugs.

Example of `appmap.tbl`

```
# appmap.tbl
#
# app#rev#host#apphost#local_setup#remote_setup#exe

# IslandWrite on host glaive
IslandWrite##glaive#sunrock#xhost+sunrock#setenv IOFFICE /usr/dist/pkgs/ioffice,v3.0.p;
setenv DISPLAY`hostname`\:0#\$IOFFICE/bin/bin.sun4/bin.ow3/IslandWrite

# IslandWrite on host lion
IslandWrite##lion##setenv IOFFICE /apps/island##\$IOFFICE/IslandWrite

# IslandWrite on solaris2 systems
IslandWrite##solaris2#rockstar#xhost +rockstar#setenv IOFFICE /usr/dist/pkgs/ioffice,v3.0.p;
setenv DISPLAY`hostname`\:0#\$IOFFICE/bin/bin.sun4/bin.ow3/IslandWrite

# IslandWrite default entry
IslandWrite######\$IOFFICE/bin/IslandWrite

sotool####su - markc; cd /export/local/sotool##./sotool -x conti

guide##-saranac####/export/local/guide/bin/guide

frame#2.0#####/export/local/frame/bin/maker

frame#3.1#####/usr/dist/exe/imaker,v3.1a

ileaf######/usr/dist/exe/ileaf

# Wingz release 3.0 on host glaive
Wingz#3.0#glaive#sunrock#xhost +sunrock#setenv DISPLAY `hostname`\:0#/usr/local/bin/Wingz
# Wingz release 3.5
Wingz#3.5#####/apps/wingz,3.5/bin/wingz
# Wingz release 3.0
Wingz#3.0#####/apps/wingz,3.0/bin/wingz
# Wingz release 3.0... default entry
Wingz######/apps/wingz,3.0/bin/wingz

123##solaris1####/apps/lotus/bin/123

123##solaris2#sunrock#xhost +sunrock#setenv DISPLAY `hostname`#/apps/lotus/bin/123
```

Guidelines for Building `appmap.tbl`

Group entries for a specific application together for ease of reading — i.e., sort by first field:

```
frame#1#solaris1###maker
frame#####maker

IslandWrite#####/path/IslandWrite
```

`rev` *Field*

If a user doesn't specify a revision on the command line (`--r` ***n***), they *only* get a match on those lines that match the application and have a blank `rev` field. Therefore, you *must* have an entry/line with a blank `rev` field. Place this line at the end of the group of lines for the application. This line acts as the default.

`host` *Field*

Remember: The first line which has acceptable `rev` and `host` field entries is used. This is extremely important when you want to deny an application to be run on a given host (i.e., `-saranac`). Place these lines first.

Lines with a valid `host` field entry apply only to that specific host and are ignored when called by other hosts. Place lines with a valid `host` field entry second, so they're tested before lines with no entry.

Place lines that specify `solaris1` and `solaris2` after lines that specify a particular host but before lines with no `host` entry. You should place lines with no `host` entry last, as they're acceptable to any system.

Defaults

You should consider having a line with both the `rev` and `host` fields empty at the end of the grouping of lines for an application. This acts as a default which would cover any system which isn't addressed by other lines.

local_setup, remote_setup, *and* exe *Fields*

The easiest way to determine what information to place in the final four fields is to start the application manually at a system and record the steps taken to do so. Example: To start IslandWrite on system sparcy, the user types:

```
prompt% setenv IOFFICE /apps/island
prompt% $IOFFICE/bin/IslandWrite
```

An appmap.tbl entry specific to sparcy would thus be:

```
IslandWrite##sparcy##setenv IOFFICE /apps/island##$IOFFICE/bin/IslandWrite
```

Note that the full path is given in the exe field. If only "IslandWrite" were used in this field, it would have created a loop, since the first occurrence of "IslandWrite" in the user's path is the link to appmap!

The final step is to create the link in APPMAP_HOME:

```
prompt% ln -s appmap IslandWrite
```

Testing

Before starting the application, you should run it in "test" mode, to see what appmap will do:

```
prompt% IslandWrite --t
```

appmap will report the value of the settings for fields 1-7, allowing you to verify that appmap has chosen the correct line in appmap.tbl.

Special Characters

In the final three fields (local_setup, remote_setup, and exe), there are considerations for specific characters and when you should or should not protect them with a backslash (\):

a. The backquote and the dollar sign:

   The backquote (`) and dollar sign ($) *should not* be protected if you want them interpreted on the local system, and *should* be protected if you want them interpreted on the remote system.

   Example: if you use `hostname` in the local_setup field, you *do not* want to protect it. If you use OPENWINHOME in the local_setup field, you *do not* want to protect it, but if you use OPENWINHOME in the remote_setup field, you *do* want to protect it.

b. The colon:

The colon (`:`) character *must* be protected by a backslash (\) in all cases.

c. The semicolon:

The semicolon (`;`) *should not* be preceded by a backslash (\) in any case.

Examples:

In the `remote_setup` field, it is often useful to set the `DISPLAY` variable to the local system name. To do so:

```
setenv DISPLAY `hostname`\:0
```

If a user's startup files does not set `OPENWINHOME`, you can set it in `appmap.tbl`. Here is how to set it in the `remote_setup` field:

```
setenv OPENWINHOME /pkgs/openwin;setenv LD_LIBRARY_PATH \$OPENWINHOME/lib
```

Notice that we did protect `$OPENWINHOME` in the setting of `LD_LIBRARY_PATH`, so that the `OPENWINHOME` variable was interpreted on the remote system and not locally.

appmap *Source Code*

```
#! /sbin/sh
#
# appmap : Application map program to aid System Administrators
#          with the transition to Solaris 2.x environment.
#
# USAGE : app [--r rev# | --t | --rt | --tr] [ arguments ...]
#
#          app : the actual application name.
#
#          --r : indicates a desired revision number.
#
#          --t : run the application in test{debug} mode.
#               Don't actually run the application but test the execution of appmap.
#
#          [--rt]|[--tr] : combination of the above revision & test.
#
#          arguments : the arguments to be passed to the application.
#
#
# The purpose of "appmap" is to make the System Administrator's life easier as they
# attempt to maintain a mixed mode environment of Solaris 2.x and Solaris 1.x machines as
# well as the appropriate application that runs in each environment.
#
# Each application is actually a link to appmap which consults an ASCII file that
# determines how a particular application will get executed. This ASCII file (appmap.tbl)
# is maintained by the System Administrator in the APPMAP_HOME directory. It is assumed
# the user has the environment variable APPMAP_HOME set to the system-wide appmap
# directory. "appmap" will first interogate appmap directory in the user's home account
# which would override the system default table. This allows a particular user the power
# to run a specific revision of an application (whether it be a previous release or some
# new beta application that is not yet released to the user community).
#
# Many Solaris 2.x applications will run natively in the Solaris 2.x environment and some
# will run in Binary Compatablity Mode. However, some applications (for whatever reason)
# have yet to be ported and will need to run from a 4.1.x server on the network using X
# to display it back to your workstation. The ASCII file appmap.tbl allows you to indicate
# which server you can as well as can't run the application from. This allows the System
# Admin to load balance the network by determining ahead of time which 4.1.x servers will
# be your X servers in this case.
```

```
# This is revision # 1 of appmap, as we have many more capabilities we would like to add.
#
# Next REV :
#    -Possible multiple names for application (i.e. interleaf,ileaf,tps)
#    -Use of netgroups to deny access to a given application.
#     Implemented using the host field - for example:
#         -!netgroup- deny access to all those in this netgroup.
#         !netgroup- specific access to all those in this netgroup.

APP=`basename $0`
HOST=`uname -n`
USAGE="Usage: $0 [ --r revision# ]"
MODE="normal"

case `uname -r | cut -d. -f1` in
     4)   OSTYPE="solaris1";;
     5)   OSTYPE="solaris2";;
     *)   echo "Not a solaris1 or solaris2 system - see system administrator.";
     exit 1;;
esac

case $# in
     0)   ;;
     *)   case $1 in
               --t) MODE="test";
                    shift;;
               --r) REV=$2;
                    shift 2;;
               --rt) MODE="test";
                    REV=$2;
                    shift 2;;
               --tr) MODE="test";
                    REV=$2;
                    shift 2;;
          esac;;
esac

ARGS=$*
```

```
# getappentry : Get application entry in table.
#
# Loop through all entries of appmap.tbl that match application requested.
# Pipe this output through nawk to determine precedence rules for match.
#
# Check to determine if a specific revision was requested.
# If the revision specified matches the entry found then
#    if local host is denied privilage to run it then
#         prompt user with error message and exit
#    if entry found that matches a specific host(ie. saranac) then
#         call executeapp to execute the application
#    if entry found that matches OS type of your current system then
#         call executeapp to execute the application
#    if entry found has nothing specified for a specific host then
#         call executeapp to execute the application
#    else only look at those entries where no revision was specified
#    if local host is denied privilage to run it then
#         prompt user with error message and exit
#    if entry found that matches a specific host (ie. host) then
#         call executeapp to execute the application
#    if entry found that matches OS type then
#         call executeapp to execute the application
# After checking list, determine :
#    If a revision was specified and no application was found and we
#     already searched both the local and system tables then
#         print error message to user to inform them that the requested
#         revision was not in table
#    else if application not found that matches and # of entries > 0 then
#         call executeapp to execute the application
#         This indicates the last entry found would be taken as the default.

getappentry( )
{
 nawk -F# "/^`basename ${APP}`#/" ${APPMAP_DIR}/appmap.tbl | nawk -F# '{
     app=$1; host=$3; apphost=$4; local_setup=$5; remote_setup=$6; exe=$7

     if (myrev && $2 == myrev)
     {
          if ($3 == "-"(myhost))
          {
               printf("\n*ERROR:\tDenied execution of '%s' on host '%s'.\n\tNotify your
                    System Administrator !\n\n", myapp, myhost)
               appfound=1
               exit 0
          }
```

```
        if ($3 == (myhost))
        {
            executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
            appfound=1
            exit 0
        }

        if ($3 == (ostype))
        {
            executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
            appfound=1
            exit 0
        }

        if (!$3)
        {
            executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
            appfound=1
            exit 0
        }

    } else {
        if (!$2 && !myrev)
        {
            if ($3 == "-"(myhost))
            {
                printf("\n*ERROR:\tDenied execution of '%s' on host '%s'.\n\tNotify
                    your System Administrator !\n\n", myapp, myhost)
                appfound=1
                exit 0
            }

            if ($3 == (myhost))
            {
                executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
                appfound=1
                exit 0
            }

            if ($3 == (ostype))
            {
                executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
                appfound=1
                exit 0
            }
        }
    }
    }
```

```
function executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
{

     if (mode == "test")
     {
          printf("\n***********************************************")
          printf("\n* TEST/DEBUG MODE *")
          printf("\n***********************************************")

printf("\napp=%s\nrev=%s\nhost=%s\napphost=%s\nlocal_setup=%s\nremote_setup=%s\nexe=%s\nmode
=%s\nmyargs=%s\n",app,myrev,host,apphost,local_setup,remote_setup,exe,mode,myargs)
     }
     else
     {
          if (!local_setup)
               {local_setup="true"}
          if (!remote_setup)
               {remote_setup="true"}
          if (apphost && myhost != apphost)
          {
               printf("\n")
               system(local_setup "; rsh " apphost " \"" remote_setup " \;
                    " exe " " myargs "\" &")
          }
          if (!apphost || myhost == apphost)
          {
               printf("\n")
               system(local_setup " \; " exe " " myargs " &")
          }
     }
}

END
{
     if (myrev && !appfound && !localcheck)
     {
          printf("\nERROR: '%s' rev '%s' not in appmap.tbl, consult your system
               administrator.\n\n", myapp, myrev)
          exit 0
     } else if (!appfound && !localcheck && NR > 0)
          {
               executeapp(app,host,apphost,local_setup,remote_setup,exe,mode,myargs)
               exit 0
          }
     else if (!appfound) exit 1
} ` myapp=$APP myrev=$REV myhost=$HOST mode=$MODE ostype=$OSTYPE myargs="$ARGS"
   localcheck=$LOCALCHECK
}
```

```
# Check if user has a local appmap directory in their home account. Remember, an
# appmap.tbl in the user's home directory overrides the system-wide default table.

APPMAP_DIR=$HOME/appmap
if [ -s $APPMAP_DIR/appmap.tbl ]; then
     # a local appmap.tbl, let's interrogate...
     LOCALCHECK=1
     getappentry

     if [ $? != 0 ]; then
          # No appmap.tbl in user's account, check system-wide appmap.tbl
          echo "Checking appmap System File"
          APPMAP_DIR=$APPMAP_HOME
          LOCALCHECK=0
          getappentry
     else
          # successful.
          exit 0
     fi
else
     # No appmap.tbl in user's account, check system-wide appmap.tbl
     echo "Checking appmap System File"
     APPMAP_DIR=$APPMAP_HOME
     LOCALCHECK=0
     getappentry
fi

if [ $? != 0 ]; then
  /usr/5bin/echo "\nERROR : '$0' not found in appmap.tbl, consult your system administrator.\n"
fi
```