AIPS MEMO NO. 66

An Overview of the AIPS TV Servers

Chris Flatters (Updated by Dean M. Schlemmer)

February 4, 1991

National Radio Astronomy Observatory
Edgemont Road, Charlottesville, VA 22903-2475
Phone: (804) 296-0211 Fax: (804) 296-0278
Internet: aipsmail@nrao.edu Bitnet: aipsmail@nrao

INTRODUCTION

This document gives an overview of the *AIPS* TV servers that are available for Unix workstations. It is oriented towards the *AIPS* manager who wishes to set up one of these servers and should be read in conjunction with the the Unix *AIPS* installation guide. If you are not familiar with *AIPS* TV concepts you should read Chapter 10 of *Going AIPS* before continuing.

The *AIPS* TV servers allow a window on a workstation screen to be used as an *AIPS* TV by an *AIPS* running on the workstation, an *AIPS* running on a remote machine or both. They currently work only under BSD Unix and other flavours of Unix systems that have BSD networking extensions (XAS should also work under VAX/VMS using mailbox communications but NRAO does not have the hardware necessary to test or maintain this option).

The following section explains the basic priciples of the TV servers.

THE BASIC OPERATION OF AN *AIPS* TV SERVER

As is the case with many networked applications, the workstation implementations of *AIPS* TV devices conform to the *client-server* model. The TV server "owns" a window on the workstation screen and is responsible for maintaining it in the workstation's windowing environment. The server must typically respond to requests to move the window on the screen, place it in front of or behind other windows, change its size, or to shrink it to an icon.

The window maintained by the server is used as an *AIPS* TV by the server's client programs (the main AIPS program and the TV tasks). The client programs may run on the workstation on which the server is running or may run on any machine that can access the server *via* a network. The clients may not access the TV window directly but must request the server to perform the TV operations and return the results, if any. This achieves two things: the client is insulated from the actual mechanics of interacting with a window on a workstation and the window is protected from two or more clients trying to make simultaneous, and possibly irreconcilable, changes to it.

In order for the server and clients to be able to communicate both the server and the client must use the same network *protocol*. The network protocol determines how messages are passed between programs. The *AIPS* TV servers support the Internet *transmission control protocol* (TCP) and an alternate protocol that provides faster communications between programs that are running on the same machine but excludes access from other machines. Both protocols are *connection oriented* which means that a connection must be established between two programs before they can send messages to each other. This is similar to the way in which a connection must be made between two telephones (at an exchange) before two people can use them to talk to one another.

In BSD Unix the two ends of a connection are known as *sockets*. A socket is completely specified by its domain (the INET domain corresponds to TCP and the UNIX domain corresponds to the faster, internal protocol), the machine on which the socket exists and a unique address. The address need only be unique on the machine on which the socket exists; it need not be unique on the network. INET domain addresses are *port numbers*, integers between 0 and 65535 while UNIX domain addresses are file pathnames.

If the client is to be able to establish a connection with the server the client must have enough information to completely specify the server's socket. In the INET domain the server is started with a prearranged port number and the machine on which the server is running is supplied to the client (via an environmental variable) at run-time. Rather than hard-coding a port number into the server and clients a symbolic *service name* is used. The mapping between service names and port numbers is defined by the file `/etc/services` on each machine. This file is normally maintained by the system manager. If Network Information Services (NIS), also known by the older name of Yellow Pages, is running then the same `/etc/services` file may define the mapping for several machines. In the UNIX domain, the server uses a prearranged pathname (usually a file in `/tmp`) and the machine on which the server is running is implicitly the same as that on which the client is running.

When a client wishes to access the TV it sends a message to the server requesting a connection. It then waits for the server to establish the connection. Once the connection has been established the client sends instructions to the server and waits for any return messages. When the interaction is complete the connection is destroyed so that another client may access the TV.

The TV server responds to a limited set of commands that fall into the two classes of display requests (*e.g.*, display a row or column of pixels) and information requests (*e.g.*, return the current cursor coordinates). It does not maintain an image catalogue: image catalogues are maintained by the clients (*via* the Y-routines).

IMAGE CATALOGUES AND TVMON

The image catalogue is an important part of the *AIPS* TV model. Each TV display should have its own image catalogue which records what images are displayed, where they are displayed on the screen, what the astronomical coordinate system is for each image and the relation between the brightness or color on the screen and physical units of brightness. This allows multiple AIPS tasks to collaborate in dealing with images on the screen.

The obvious consequence of having the Y-routines handle the image catalogue rather than the TV server is that the image catalogue resides on the same machine as *AIPS* itself. This is not a problem if *AIPS* and the TV server are both on the same machine (*e.g.*, a standalone workstation) since there is only one display and one image catalogue. Similarly it is not a problem if *AIPS* resides on one machine and the TV server resides on only one remote workstation. The case where there are several workstations is somewhat more complicated since each workstation must be assigned to a different TV number in order to have its own catalogue (although this becomes unwieldy if there are many workstations).

The real problem occurs if the same workstation acts as the display for copies of *AIPS* running on several machines. In this case each copy of *AIPS* has its own separate image catalogue despite the fact that each of these catalogues refers to the same device. Consequently the image catalogues can easily get out of step with respect to what is displayed on the screen if any two (or more) *AIPS* try to share the display. This is not a critical problem but it is clearly undesirable.

The TVMON program is intended to solve this problem. TVMON is an *AIPS* program that sits between the *AIPS* clients and a TV server. TVMON usually resides on the same machine as the server and handles all the Y-routine instructions from the client, including those that address the image catalogue. TVMON, therefore maintains the image catalogue. It may also be used to make the more old-fashioned kind of display (represented by the I^2S) available over a network. The disadvantages of TVMON are that it is a FORTRAN program that must be linked with the *AIPS* libraries and that it requires some part of *AIPS* to be running on the same machine (in order to create and maintain image catalogues). You can not, therefore, use TVMON if your display workstation has no FORTRAN compiler or has insufficient diskspace to install a minimal subset of *AIPS* (you will need at least 200 Megabytes of disk space for a minimal AIPS and between 300 and 350 Megabytes for a full installation; if sharable libraries can be used the full installation will shrink to about 200 Megabytes).

If, however, you have a FORTRAN compiler on your display workstation and sufficient disk space to install *AIPS* it is strongly recommended that you install both *AIPS* and TVMON. Among other things you will find that operations that require feedback through the cursor, such as TVFIDDLE and TVPSEUDO, are much faster if they are done using an *AIPS* local to the display workstation than using a remote *AIPS* because there is a significant overhead involved in sending messages back and forth over the network. This mode of operation is impossible without TVMON.

INSTALLING AND CONFIGURING *AIPS* WITH TVMON

There are two steps to installing and configuring *AIPS* with TVMON. First, you must install *AIPS* on the client machines in such a way that it can talk to TVMON. You must then install AIPS and TVMON on the display workstation, configured appropriately for the server or TV device you are using. Before installing either *AIPS*, however there are some preliminaries which will be explained following the explanation of the various window system servers below.

THE WINDOW SYSTEM SERVERS

We shall now briefly describe the three window system servers currently supported by *AIPS*. The server you will use will depend on which window system which you are currently running on your computers and how many options you wish to utilize.

SSS — The SunView Screen Server

If you are currently using the Sunview window system (Sun Microsystems' proprietary, kernel-based windowing system for Sun Workstations) on your computer, then you MUST use the SunView Screen Server (SSS).

SSS is the oldest of the *AIPS* screen servers. It was originally written by Brian Glendenning, who is currently at the National Radio Astronomy Observatory. SSS emulates a TV with a width of 1142 pixels and a height of 803 pixels. It provides 2 image planes and 4 graphics overlay planes. It requires an 8-bit color or grey-scale display and can display up to 112 colors or levels of grey-scale. The size of the TV and the number of grey-scale planes (up to a maximum of 4) may be changed by modifying `#define` statements in the file `header.h`.

The *AIPS* buttons A through D are tied to the function keys F3 through F6 on the keyboard. Function keys F2 and F7 may be used to switch between a large and small display window. The larger window is fixed at the maximum size of the TV while the smaller window may be resized by the user from the SunView window menu.

Unfortunately SunView only allows one cursor to be displayed and this must be the window system cursor. This means that there is no visible *AIPS* TV cursor. However, pressing or holding down the left-hand mouse button while the cursor is in the TV window forces the *AIPS* cursor to the same position as the window system cursor.

XAS — The X Window System *AIPS* Server

If you are currently using the X-Window windowing system on your computer, you may use either this screen server or the XVSS server system (described in the next section). This system is not as versatile as XVSS, but is simpler, faster, and does not require an X toolkit. XAS will ONLY run on displays that are 8-bits deep. If you do have X-Windows installed, read BOTH this and the next section on XVSS before deciding on which server to use.

XAS is a generic *AIPS* TV server for the X Window System originally written by Tom Pauls and Ralph Gaum at the Naval Research Labs in Washington. It uses the low-level Xlib library but does not use any X toolkit.

XAS emulates a TV with a width of 1024 pixels and a height of 720 pixels and provides 2 image planes and one graphics overlay plane. It requires an 8-bit color display with a pseudocolor visual and can display up to 64 different colors. The TV size and the number of grey-scale planes may be changed by modifying **`#define`** statements in **`xas.h`**.

Like SSS, XAS maps the *AIPS* buttons to function keys F3 through F6. Unlike SSS, however, there is no size toggle. Users may resize the window in any way appropriate to their preferred window manager. Cursor handling is identical to that of SSS.

XAS will work with any window manager that complies with the inter-client communications conventions manual (ICCCM) recommendations. Examples are **`twm`** (X11R4 release), **`olwm`** and **`mwm`**.

XVSS — The XView Screen Server

If you are currently using the X-Window windowing system on your computer, you may use this server. Although some operations may be slower, XVSS is more versatile than XAS. However, you will also need the XView toolkit available for installation; Otherwise, you MUST use XAS. Here is some background:

Like XAS, XVSS is an X Window System based screen server. UNLIKE XAS, it uses an X toolkit. The particular toolkit used here is the XView toolkit from Sun Microsystems. XView is freely available in source form and will run on most BSD compatible Unix systems. XVSS requires release 2.0 or later of the XView toolkit, and this release is available *via* anonymous ftp from `expo.lcs.mit.edu` (18.30.0.212) and requires 15 to 20 Mbytes of disk space to build.

XVSS provides a screen size of 1024 by 720 pixels with 2 image planes and one graphics overlay. The display size and number of image planes (up to 4) may be changed by modifying `#define` statements in `header.h`. XVSS is the only screen server that will run on displays that are not 8-bits deep. By default XVSS is configured to provide 63 but can be easily modified to display more colors if it is used on a screen that can display a total of more than 256. The maximum number of colors available to XVSS can be calculated by dividing the maximum number of colors on the display by 4 (*e.g.*, 64 = 256/4). The ability of XVSS to run on machines with displays that use an unusual number of bit-planes results in it being somewhat slower than XAS on a 256-color display if *AIPS* resides on the same machine as the server (the typical ration of speeds is 2/3). If used remotely the speeds of both servers are dominated by network I/O and are consequently almost identical.

XVSS has identical button assignments and cursor handling to SSS. However, it also provides a control panel with a resize button and buttons A, B, C and D. The user may click on any of these buttons as an alternative to using the function keys.

XVSS allows the user (or, more often, the AIPS manager) to choose the color of the graphics overlay at run time. This is not currently possible for EITHER of the other two servers. It also has on-line help available and can take advantage of shared memory to speed up image transfer operations if it is available. These special features of XVSS will be described following the instructions for installing a screen server.

XVSS will work with any ICCCM compliant window manager that allows colormaps to be associated with subwindows using the XA_WM_COLORMAP_WINDOWS property (examples are `olwm` and `mwm`). It will normally work with other ICCCM compliant window managers (*e.g.*, `twm`) provided that there are at least 128 free entries in the default colormap, depending on the properties of the X Window server.

Which Server Should I Use?

Clearly, if you have a Sun Workstation WITHOUT X-Windows, you have no choice but to use SSS.

Of the two X Window System based servers, most users will prefer XVSS, since the visible control panel makes it easier to use (at least for beginners). However, you cannot run it if you do not have XView available (or cannot install it for some reason).

If you need to run TVMOVIE, XAS might be preferred anyway because of its greater speed. However, if you have an unusual number of bit-planes (other than 8), then you cannot use XAS.

With these criteria, you should now be able to narrow your choice of TV servers to one. The rest of this document is broken into three basic sets of instruction (one for each server), and you need only read those sections which pertain to the server which is correct for your configuration. If you are using a single, stand-alone workstation, SKIP all sections pertaining to CLIENTS only.

PRELIMINARIES

Before installing *AIPS* (on either the display or client machines), there are some preliminary steps to be performed. These are listed below; Subsection titles indicate which machine(s) these steps are to be performed on; Skip those steps which do not apply.

The following section applies to clients and servers ONLY if you are networking between the two. If you have a stand-alone system, SKIP this section on modifications to `/etc/services` ENTIRELY:

Modify /etc/services (clients AND servers):

TVMON only operates in the INET domain. It therefore requires its port number to be known both on the display workstation and on all possible client machines. This is done by adding the line

```
VTVIN           5001/tcp
```

to the file **/etc/services** on EACH MACHINE (all clients AND servers).

NOTE: If you have network information services (NIS — formerly known as yellow pages) running on your system, you need only modify **/etc/services** on the NIS host machine. The number is arbitrary but must be greater than or equal to 5000 and must not exceed 65535. It is recommended that you use 5001 for compatibility with other *AIPS* sites; this way you may use your workstation to display images from a remote site via Internet or a remote user may display images from your machine.

Normally the superuser is the only person who may change **/etc/services** so you may have to get your system manager to make the changes.

If you are NOT using TVMON, you will also need to add the line:

```
SSSIN           5000/tcp
```

to the **/etc/services** file, if you are using either SSS or XVSS , or alternately the line:

```
XASIN           5000/tcp
```

if you are using XAS.

Client Machines ONLY:

Before compiling *AIPS* on the clients, you should modify the file `$SYSLOCAL/LIBR.DAT` so that *AIPS* is linked with the Y-routines in the `$YVTV` area. For example, assuming that the remote server display is the only display available or is TV device 1, `LIBR.DAT` should contain the following specification for the Y-routine library:

```
Virtual TV Y-routines

$LIBR/YVTV/SUBLIB:0:$YVTV
$LIBR/YVTV/SUBLIB:0:$YGEN
```

[Note: having `:0:` insures that the executables will be in the area `$LOAD`. However, if you are going to have, say, one ADDITIONAL display device (for a total of TWO alternates), you would ADD ANOTHER set of lines like the above two, instead replacing the "`:0:`" with "`:2:`", and also `/YVTV/` and `$YVTV` with appropriate directory and logical names; This will place THOSE executables in the area `$LOAD/ALT2`. If you require even more display devices (probably unlikely), `:#:` can be increased further (again, see the full installation guide for more details).]

All the programs in the areas `$AIPPGM`, `$QYPGM`, `$QYPGNOT`, `$YPGM`, and `$YPGNOT` must also be linked to the alternate version of *AIPS*. Look through `LIBR.DAT` for the sections containing the links to these areas and add appropriate versions of the `$LIBR/YVTV/SUBLIB` line. For example, under the `$AIPPGM` area, add the line `$LIBR/YVTV/SUBLIB:0:$AIPPGM`, and so on for the rest of the specified areas.

Also, be sure that your version of `LIBR.DAT` does NOT include a linkage specification for area `$YPGVDEV` (the area containing the virtual TV server programs); Comment it out if it does exist, as it is reserved only for the server.

You may then compile and link *AIPS* on the client machines as described in the installation guide.

In order to use TVMON, you must also define the *AIPS* logical name of the TV device. For safety, this definition must be placed in BOTH `$SYSLOCAL/ASSNLOCAL.???` (where ??? is `SH` or `CSH`) and `$SYSLOCAL/AIPS`. If there is not a copy of `AIPS` in the `$SYSLOCAL` area, copy one from the `$SYSUNIX` area and then edit it. The line to be added is `VTVIN:`*machine*, where *machine* specifies the display machine. *Machine* can be either an Internet name (*e.g.*, cholla.aoc.nrao.edu), an abbreviated name (*e.g.*, cholla), or an Internet address (*e.g.*, 192.43.204.2).

As an example, IF YOU ARE USING THE BOURNE-SHELL and the remote display is to be called TV device 1, `ASSNLOCAL.SH` and the *AIPS* start-up script `AIPS` should contain the lines:

```
TVDEV1=VTVIN:mydisplay
export TVDEV1
```

If, however, you are using the c-shell, you need only add the line:

```
setenv TVDEV1 VTVIN:mydisplay
```

In either of these cases, the TV device will be defined automatically at login.

The above step is sufficient for your TV logical definition. If, however, you want a more flexible way to choose between multiple possible displays, you can have the user type in the machine he wants at login. If you would like to have this option AND ARE USING THE BOURNE-SHELL, replace the first line in the two BOURNE specifications above with:

```
echo "Enter the name of the display machine"
read name
TVDEV1=VTVIN:$name
```

in the start-up script AIPS. The user will then be prompted for the device at login. (Note, however, that some extra shell programming will be necessary to avoid having the user crash *AIPS* if he mistypes the machine name.)

NOTE that you must give TVDEV1 a value in ASSNLOCAL.SH, even if you are going to change it later.

Server Machines ONLY:

On the server, you must configure the file $SYSLOCAL/LIBR.DAT which will be used to generate and link the appropriate Y-routines for the device you will be using. Look for the area $YPGVDEV (one of the areas for Y-routine link specifications) in the program section of LIBR.DAT and be sure it includes the Y-routine library linkage specification which is correct for your server system. For example, if you are using the SunView Screen Server, you should have the following line in the $YPGVDEV section of LIBR.DAT:

```
$LIBR/YSS/SUBLIB:0:YPGVDEV
```

If you are using one of the other systems, replace YSS in the line above with either XAS or XVSS. This will ensure that TVMON is built when AIPS is compiled.

You must also provide a means of starting TVMON with the appropriate logicals defined. The following Bourne shell script will start TVMON; The following lines must be added at either the end of the *AIPS* start-up script $SYSLOCAL/AIPS;

```
VTVDEV1=VTVIN:dummy          # it doesn't matter what comes after the colon
export VTVDEV1
TVDEV1=/tmp/aips_screen          # for SSS (see below)
export TVDEV1
ps -ax | grep TVMON | grep -v grep > /dev/null
if test "$?" = "1"
then $LOAD/TVMON.EXE &
fi
```

IF YOU HAVE NO CLIENT MACINES, replace all references to TVMON in the lines above with the name of the screen server you are using (SSS, XAS, or XVSS). See the next section for more details.

This is all that is necessary for the preliminaries.

INSTALL THE SCREEN SERVER

Installation of the three servers consists of the same steps for each, although they differ in minor detail. In the following example we will describe the installation of XVSS and note any differences for for the installation of SSS and XAS in italic font.

Configure the file LIBR.DAT (server AND clients):

The file `$SYSLOCAL/LIBR.DAT` must be configured to build the necessary Y-routine library and to link the *AIPS* tasks with it. This step should be done (as indicated in the AIPS Unix Installation Summary) *prior* to running INSTEP2 of the UNIX *AIPS* installation. If the following lines are not already in `$SYSLOCAL/LIBR.DAT`, add them (these define the libraries):

```
$LIBR/YSS/SUBLIB:0:$YSS
$LIBR/YSS/SUBLIB:0:$YGEN
```

These two lines are sufficient if you are running EITHER `SSS` or `XVSS`.

NOTE: if you are using XAS, use `$YXAS` *instead of* `$YSS` *in the first line above.*

You may then install *AIPS* as normal.

Build the Screen Server (server machine ONLY):

The source code for the screen servers is packed into three `.SHR` (which stands for SHell aRchive) files in the `$YSERV` area. The source codes for `XVSS`, `XAS`, and `SSS` are packed into the files `XVSS.SHR`, `XAS.SHR`, and `SSS.SHR`, respectively. First, move to the `$YSERV` area and create a directory in which to build the screen server you will be using (use obvious directory names like `SSS`, `XAS`, or `XVSS`). Then, copy the appropriate `.SHR` file into it . (NOTE: If you are building XVSS, you will also want to copy the file `$YSERV/XVSS.UU` to `xvss.uu` in the same subdirectory; It contains the help data for XVSS in an encoded form. The source code files must then be "unpacked"; On a Unix system, simply type:

`sh XVSS.SHR` (or `sh XAS.SHR` or `SSS.SHR`)

You will then end up with a variety of files in that directory. [We have noticed that at the beginning of one of the files created during the "unpack" (`header.h`), an occasional spurious character(s) may appear; Check this file and edit out any offending characters.] FOR NON-UNIX SYSTEMS, you will also have to copy the program `UNSHR.FOR` from the `$YSERV` area, compile it, then run it.

Then you will end up with various C source files and a `Makefile` in that area. You may want to inspect the compile-time constants in the `.h` files and modify them for your system - however, it is NOT recommended that you do this UNLESS you are sure that you know what you are doing.

While you are still in `$YSERV/yourserver`, you should then edit `Makefile`. You will probably have to redefine some macros at the top of the file:

1.) FOR ALL THREE SCREEN SERVERS: `FLAGS` is used to give any other flags which might be needed by the C compiler you are using. If you are compiling on a Sun 3, this is where you put the floating-point option, *e.g.*,

```
FLAGS = -f68881
```

2.) FOR XVSS SERVERS ONLY: `DESTDIR` is the destination directory for the XVSS binaries (this should be the same as the area $LOAD if you have installed *AIPS* on the display machine).

3.) FOR THE XVSS SERVER, the libraries you may need to look for are `libxview.a` and `libX11.a`. If, for example, `libX11.a` is in directory `/usr/X11/R4/lib` and `libxview.a` is in `/Openwins/lib`, you should modify the `LIBDIRS` definition in the `Makefile` to look like this:

```
LIBDIRS = -L/usr/X11/R4/lib -L/Openwins/lib
```

4.) FOR XAS AND XVSS SERVERS ONLY: `LIBDIRS` gives the location of any libraries that are not in the standard directory `/usr/lib`; Each directory in the `LIBDIRS` definition should be preceded by `-L` (there will likely be some `LIBDIRS` macro entry already there as an example).

The other macro which may need modification to run under `XAS` or `XVSS` is `INCDIRS`. `INCDIRS` is similar to `LIBDIRS`, but it gives the locations of any standard header files that are not in their standard location (`/usr/include`). Precede each directory with -I. An example might be:

```
INCDIRS = -I/usr/X11/R4/include -I/Openwins/include
```

5.) FOR XVSS ONLY: `HELPDIR` gives the directory in which the on-line help data will be installed, and `SHMOPT` enables the use of the shared memory extension. (See the comments in the makefile and the section on the special features of XVSS for more information.)

Once all macros are correctly defined, you may then compile the server by typing '`make`'. If the compile fails, this usually indicates that the above variables have been set incorrectly. Correct them and try again.

Build a Shell Script to Start the Screen Server

This section gives some brief guidelines on assigning environment variables and path(s) to start up the screen servers. Read this section before configuring the startup scripts described in the next section.

FOR SSS AND XVSS SERVERS ONLY:

For these two screen servers, the environment variables `TVDEV` and `TVDEV`*n* MUST be set BEFORE the server is started (here, *n* is the TV device number). In your case, TVDEV must be DEFINED AS TVDEV*n*. This is best done via a shell script which will automatically set the TV variables and then start the server in the background. If you are using TVMON, you will probably want to start both TVMON AND the server from the same script.

If you ARE NOT using the server over a network (*i.e.*, ARE NOT using TVMON), TVDEV*n* should be set to an absolute pathname (*i.e.*, one beginning with a /); we recommend that you specify a pathname in `/tmp`, for example `/tmp/aips_screen`.

If you ARE using the server over a network but ARE NOT using TVMON, `TVDEV`*n* should be defined as `SSSINB:`*machine*, where the B suffix allows certain operations to be buffered for efficiency. Here *machine* is the display (server) machine.

If you ARE using TVMON AND the server OVER THE NETWORK (the usual case), give `TVDEV`*n* an absolute pathname like `/tmp/aips_screen` as described above.

You may use a startup script similar to that shown earlier for TVMON (see next section for sample).

XAS ONLY:

The environment variable `AIPSTV` must be set before the server is started. When using XAS, *n* MUST be 1. This is best done via a shell script that will automatically set the TV variables and then start the server in the background. If you are using TVMON you will probably want to start both TVMON and the server from the same script. Enter the lines:

```
TVDEV=TVDEV1
AIPSTV=SSSIN
TVDEV1=SSSIN
```

at end of the *AIPS* startup script (`$SYSLOCAL/AIPS`).

If you ARE using the server over the network but ARE NOT using TVMON, you should set `AIPSTV` AND `TVDEV1` to `SSSIN`.

If you are NOT using the server over a network, then you should set `AIPSTV1` AND `TVDEV1` to an absolute pathname (*i.e.*, one beginning with a /); we recommend that you specify a pathname in `tmp`, for example `/tmp/aips_screen`.

You may use a startup script similar to that shown earlier for TVMON (see next section for sample).

Configure ASSNLOCAL.SH or the *AIPS* Startup Script

First, TVDEV*n* MUST be given a value in ASSNLOCAL.SH (see guidelines above); You may change it later in the AIPS script if you wish.

Next are two sample scripts for c-shell or Bourne-shell, whichever you are using. you can either add the lines at the end of the *AIPS* startup script (AIPS) or the ASSNLOCAL.SH or ASSNLOCAL.CSH files in $SYSLOCAL:

For the c-shell:

```
#!/bin/csh
ps ax | grep SSS | grep -v grep >/dev/null
if ($status == 1) then
  echo -n "Sun Screen Server not running.  Start it? (y/n) "
  set yn = ($<)
  if ($yn == 'y') then
    $LOAD/SSS &
  endif
endif
```

And for the Bourne-shell:

```
#!/bin/sh
ps ax | grep SSS | grep -v grep >/dev/null
if [ "$?" = "1" ] ; then
  echo -n "Sun Screen Server not running. Start it? (y/n) "
  read yn
  if [ "$yn" = "y" ] ; then
    $LOAD/SSS &
  fi
fi
```

Obviously, SSS would need to be replaced by XAS or XVSS if you were using one of the other servers.

RUN SETTVP

Now run the program SETTVP; Type:

```
RUN SETTVP
```

anywhere.

The following table gives the appropriate SETTVP values for each server as they are distributed. If you modify the configuration parameters in the code you should revise the SETTVP parameters to match.

Parameter	SSS	XAS	XVSS
No. of grey-scale planes	2	2	2
No. of graphics planes	4	1	1
Images per plane	256	256	256
Size of planes	1142, 803	1024, 720	1024, 720
Max. grey scale intensity	111	63	63
Peak intensity out of LUT	255	255	255
Peak intensity in/out of OFM	255, 255	255, 255	255, 255
X, Y min. scroll increments	1, 1	1, 1	1, 1
Maximum zoom factor	-15	-15	-15
Type of split screen allowed	0	0	0
X-axis write mode	3	3	3
Y-axis write mode	3	3	3
Number of ALUs	0	0	0
Number of ISUs	0	0	0

This completes the installation.

SPECIAL FEATURES OF XVSS

This section describes features of XVSS that are not currently available in the other screen servers.

Graphics Plane Colour

By default, XVSS displays overlayed graphics in green. It may be desirable to change this for a variety of reasons. As an example some Sun 3 workstations are equipped with greyscale displays that are run from the red signal of the display controller so that green is not visible.

XVSS takes advantage of the X resource database to allow the user to specify the graphics overlay color. Different workstations may share the same XVSS executable but still display graphics using different colors. The resource database is read when XVSS is started; future versions may allow the graphics color to be modified from a properties window while XVSS is running.

XVSS takes the graphics overlay color from the value of the **xvss.graphicsColor** resource. The value may be any valid X window color specification: either a named color from the color names database or a hexadecimal RGB specification. The X resource database is maintained in the X server and is initialised from the **.Xdefaults** file when the X Window System is started. It may be later changed by the **xrdb** command. See your local X Window System documentation for further details.

On-Line Help

XVSS interfaces to the OPEN LOOK help system. In order to use this you must set the search path in the **HELPPATH** environmental variable to include the directory in which you installed the file **xvss.info**. This is best done as part of the startup script. Help is accessed by pressing the help key (normally bound to F1 on the main keyboard).

Shared Memory

XVSS supports the MIT shared memory extension to the X Window System. Your operating system must be configured to support System V shared memory facilities in order to use this and you must compile XVSS with the shared memory option set in the makefile.

The shared memory extension should speed up some XVSS operations but may cause severe paging problems on machines with small memories. Do not try to use it if you do not have at least 16 Mbytes of physical memory; however, since the use of shared memory can be turned off and on, it does no harm to compile with the shared memory option enabled if you have System V shared memory (if you don't you will get link errors when you build XVSS).

Shared memory is controlled using the X resource database. On startup XVSS reads the boolean resource **xvss.useSharedMemory**: if this is true then the shared memory extension is used. If **xvss.SharedMemory** is false or is missing then the shared memory extension will not be used. XVSS will indicate whether or not is using shared memory in the window footer if you are using an OPEN LOOK window manager.

Not all vendors support the MIT shared memory extension. Use the **xdpyinfo** command and look for the identification string "MIT-SHM" to see if it is available. SunOS may be configured to support shared memory.