AIPS MEMO NO.

An Overview of the AIPS TV Servers

Chris Flatters

February 4, 1991

Revised by: Dean M. Schlemmer, Pat Murphy, Eric Greisen, Mark Calabretta

National Radio Astronomy Observatory
Edgemont Road, Charlottesville, VA 22903–2475
804–296–0211 (FTS 940–7211), x328; TWX 910–997–0174
aipsmail@nrao.edu

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 principles 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. The "TV display" is in fact a creation of the software of the server. The server structure is required in order to have the TV window act like a device and be accessible, in sequence, to more than one *AIPS* task.

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 catalog: image catalogs are maintained by the clients (*via* the Y-routines).

IMAGE CATALOGS AND TVMON

The image catalog is an important part of the *AIPS* TV model. Each TV display should have its own image catalog 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 catalog rather than the TV server is that the image catalog 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 catalog. 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 catalog (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* could have its own separate image catalog, despite the fact that each of these catalogs refers to the same device. Consequently the image catalogs 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. Network File System (NFS) can be used to handle this problem. All relevant machines should mount a common disk partition (*e.g.*, /AIPS). Then, each host would be given its system-file area (DA00) below this common area (*e.g.*, /AIPS/DA00/$HOST, meaning /AIPS/DA00/ATELES for host ateles and /AIPS/DA00/CARINA for host carina). Then, within each host-specific area, the IC and ID (and also PW and GR) system files are hard linked to all other host system areas. There is therefore only one copy of each IC and ID file for a particular server, but with many aliases. Note, that this solution will have to be implemented by the local AIPS Manager; provisions are not made for it on the NRAO-provided installation tape.

The TVMON program is another solution to this problem and to the problem of more than one kind of TV device on a single host. TVMON is an *AIPS* program — another server — that sits between the *AIPS* clients and a TV server. TVMON usually resides on the same machine as the server, although it does not have to, and handles all the Y-routine instructions from the client, including those that address the image catalog. TVMON, therefore maintains the image catalog. 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 catalogs). You cannot, 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 around 30 Megabytes of disk space for a minimal *AIPS* plus additional temporary space used only during installation. A full installation of *AIPS* requires more like 200 Megabytes.

If, however, you have a FORTRAN compiler on your display workstation and sufficient disk space to install *AIPS*, you should consider installing both *AIPS* and TVMON. Among other things, you will find that operations that require feedback through the cursor, such as TVFIDDLE and CURVALUE, are much faster if they are done using an *AIPS* local to the display workstation than if they are done using a remote *AIPS*. This is due to the significant overhead involved in sending messages back and forth over the network. This mode of operation is made possible most simply by TVMON, although it may be implemented in other ways as well.

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, 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, and has been substantially revised by Eric Greisen, also of the NRAO. SSS emulates a TV with a width of 1142 pixels and a height of $\leq$ 876 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 display size and number of channels may be changed by changing the `#define`s in the file `header.h`. Parameter `SCREEN_WIDTH` must be $\leq$ 1142 and an even number. Parameter `TEXT_SPACE` must be $\geq$ 0 and an even number; the screen height is given by 876− `TEXT_SPACE`. Parameter `NGREY` must be $\leq$ 4. Note that a megabyte of memory is required for each channel, so use more than 2 only on large-memory machines. Two channels are needed for blinking. Do not change parameter `NGRAPH`; no speed or memory advantage may be obtained thereby.

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 and moved by the user using the usual SunView window procedures.

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. The cursor will change to a purple color, however, whenever it is in the TV window.

XAS — The X Window System *AIPS* Server

If you are currently using the X-Window windowing system on your computer, you may use either the XAS or the XVSS server system. The XAS system is constructed with the most basic tools of the X-Windowing system. It lacks, therefore, useful "panels" and "buttons," but it should run on any X-Windowing system without requiring special X software "toolkits." It should run on displays that are not 8-bits deep, and even on VMS systems, but the testing of these options is not so complete. XVSS also supports systems which are not 8-bits deep, but requires the XView toolkit and makes no pretense of running under VMS (or even AIX). 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 and does not use any X toolkit. It has been substantially revised and upgraded by Eric Greisen.

XAS emulates a TV with a width which adapts to the workstation on which it runs. When it begins, it announces (on "stderr") the screen dimensions it will use and the maximum pixel intensity it can accept. These parameters must then be entered in the TV parameter file by the local AIPS Manager using the SETTVP program. On an 8-bit SUN, these are 1142 by 800 with pixels values from 0 through 199. On an IBM RS 6000, they are 1256 by 910 with the same range of pixel values. As delivered, XAS provides 2 image planes and 4 graphics overlay planes. It requires at least a 6-bit color display with a pseudocolor visual. The TV size and the number of grey-scale planes may be changed by modifying `#define` statements in `xas.h`. The screen height is intended to be modified with the parameter `TEXT_SPACE` which defines a space for a background text window to be visible below the TV window. The value of `TEXT_SPACE` is $\geq$ 0 and a value of 68 was assumed in the dimensions quoted above. `SCREEN_RIGHT` may also be increased if needed to provide space to the right of the TV window. Parameter `NGREY` defines the number of TV channels and has no limit other than being > 0. However, please note that a megabyte of memory is required for each channel, so use more than 2 only on large-memory machines. Two channels are needed for blinking. Do not change parameter `NGRAPH`; no speed or memory advantage may be obtained thereby.

Like SSS, XAS maps the *AIPS* buttons to function keys F3 through F6. Also like SSS, XAS uses function keys F2 and F7 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 and moved by the user using the usual X-Window procedures. XAS also uses function keys F8 and F9 to toggle two different debug flags. If you accidentally hit one of these, hit it again to turn off the flood of messages. They are of interest to those curious about what is going on "behind the scenes."

XAS should 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 216 free entries in the default colormap, depending on the properties of the X Window server. To compile XAS on VMS systems, comment out the `#define` for `BSD` and enable the one for `VMS`. For `AIX` systems, simply enable the `#define` for `AIX` leaving `BSD` also enabled. Note that XAS is quite demanding in its use of the colormap, requiring $2^{depth} - 40$ colors. It is prepared to use a virtual color map if needed to get these, but this has the effect of having all color maps switch whenever the cursor enters/leaves the XAS window. We recommend, for *AIPS* systems, that you pre-allocate as few of the entries in the default colormap as possible to allow XAS to use the default colormap.

XAS allows the user to configure certain parameters to suit him/her-self. This is done using the `.Xdefaults` file in the home (login) directory. Other files may also be used, but then the program `xrdb` must be run to enter the defaults into the X Window Manager. These parameters control the initial size and location of the TV window, the location of the icon, the color and shape of the cursor, and the colors of the graphics overlay planes. The parameters read by XAS are

`AIPStv*geometry:`	518x518+0+0
`AIPStv*iconGeometry:`	-0+0
`AIPStv*cursorShape:`	34
`AIPStv*cursorR:`	255
`AIPStv*cursorG:`	0
`AIPStv*cursorB:`	255
`AIPStv*graphics1R:`	255
`AIPStv*graphics1G:`	255
`AIPStv*graphics1B:`	0
`AIPStv*graphics2R:`	16
`AIPStv*graphics2G:`	255
`AIPStv*graphics2B:`	0
`AIPStv*graphics3R:`	255
`AIPStv*graphics3G:`	171
`AIPStv*graphics3B:`	255
`AIPStv*graphics4R:`	0
`AIPStv*graphics4G:`	0
`AIPStv*graphics4B:`	0

where the names of things are obvious, the case is important, and the values shown here are the defaults. Colors must be between 0 and 255 for the cursor and graphics channels 1, 2, and 3. Graphics channel 4 is used as a background and is limited to 0 through 63. Control of the colors may be particularly important for workstations with black-and-white monitors. In such configurations, some color values may not be displayed adequately.

The cursor shape numbers are defined in the *Xlib Reference Manual* (Volume Two of *The Definitive Guides to the X Window System*). Even numbers from 0 through 154 are legal, but not all are desirable. Possibilities include 30 (a cross with 2 lines in each direction), 40 (a square with a central dot), 128 (an ellipse with a central dot), 132 (arrow like the default cursor), and many others.

The `geometry` and `iconGeometry` codes require more explanation. The string is $< width >$ `x` $< height > [\pm]xoff[\pm]yoff$ where + offsets refer to the top and left and − offsets refer to the bottom and right. The default icon position is the top right corner of the screen and the default window position is the top left corner. Note that the width and height of the icon are set by its designer and cannot be changed. They may be omitted. The width and height set for the display window are just those which it will take at the beginning. You may resize and move the window, and move the icon, afterwards. XAS also allows the icon location and the window geometry to be set on the command line that starts XAS. It accepts `-IC` or `-IG` as advisory, overridden by the user's `iconGeometry`, if present, and `-ic` or `-ig` as overriding even the user's `iconGeometry`. Similarly, XAS takes `-G` as advisory and `-g` as definite to specify the initial window geometry. It is not a good idea to give any other command-line options.

XVSS — The XView Screen Server

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. If you do not have the XView toolkit installed, you *must* use XAS.

XVSS was originally converted from SSS and improved by Chris Flatters. It has since been substantially revised by Eric Greisen. It provides 2 image planes and 4 graphics planes. The screen size and the display depth are determined from the display itself. This is similar to XAS. Like SSS and XAS, you may alter the number of channels and the actual display size used by modifying `#define` statements in `header.h`. Parameter `NGREY` defines the number of TV channels and has no limit other than being > 0. Note that a megabyte of memory is required for each channel, so use more than the 2 needed for blinking only on large-memory machines. Do not change parameter `NGRAPH`; no speed or memory advantage may be obtained thereby. The screen height is intended to be modified with the parameter `TEXT_SPACE` which defines a space for a background text window to be visible below the TV window. The value of `TEXT_SPACE` is ≥ 0. `SCREEN_RIGHT` may also be increased if needed to provide space to the right of the TV window. Parameters `SCREEN_LEFT` and `SCREEN_TOP` may also be used to control the space around the TV window. They should be large enough to allow for the window left and right edges plus the header line and footer (if any). Space for the button panel is managed internally. They can be set still larger if the TV display is not intended to occupy the full screen.

The parameter values to be entered into the *AIPS* TV control files can be determined from the setting of the `#define`s for `NGREY` and `NGRAPH` and from a line displayed on `stderr` when XVSS starts up. That line shows the x and y dimensions of the TV display and the maximum grey level.

XVSS has identical button assignments and cursor handling to XAS, including the debug buttons F8 and F9. 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 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.

XVSS allows the user to configure certain parameters to suite him/her-self. This is done using the `.Xdefaults` file in the home (login) directory. Other files may also be used, but then the program `xrdb` must be run to enter the defaults into the X Window Manager. These parameters control the color and shape of the cursor and the colors of the graphics overlay planes. The parameters read by XVSS are

`AIPStv*cursorShape:`	34
`AIPStv*cursorR:`	255
`AIPStv*cursorG:`	0
`AIPStv*cursorB:`	255
`AIPStv*graphics1R:`	255
`AIPStv*graphics1G:`	255
`AIPStv*graphics1B:`	0
`AIPStv*graphics2R:`	16
`AIPStv*graphics2G:`	255
`AIPStv*graphics2B:`	0
`AIPStv*graphics3R:`	255
`AIPStv*graphics3G:`	171
`AIPStv*graphics3B:`	255
`AIPStv*graphics4R:`	0
`AIPStv*graphics4G:`	0
`AIPStv*graphics4B:`	0

where the names of things are obvious, the case is important, and the values shown here are the defaults. Colors must be between 0 and 255 for the cursor and graphics channels 1, 2, and 3. Graphics channel 4 is used as a background and is limited to 0 through 63. Control of the colors may be particularly important for workstations with black-and-white monitors. In such configurations, some color values may not be displayed adequately.

The cursor shape numbers are defined in the *Xlib Reference Manual* (Volume Two of *The Definitive Guides to the X Window System*). Even numbers from 0 through 154 are legal, but not all are desirable. Possibilities include 30 (a cross with 2 lines in each direction), 40 (a square with a central dot), 128 (an ellipse with a central dot), 132 (arrow like the default cursor) and many others.

XVSS does not directly handle any command-line arguments. However, it passes any arguments given to the window manager (olwm) in some of the window-creating subroutine calls.

Which Server Should I Use?

Clearly, if you have a Sun Workstation *without* X-Windows, you have no choice but to use SSS. Older Sun Workstations (*e.g.*, Sun 3s) may not have the muscle to run X Windows well, again suggesting SSS. Of the two X Window System based servers, some users will prefer XVSS, since the visible control panel makes it easier to use (at least for beginners). XVSS also uses fewer colors which may allow it to use the default colormap, when XAS must use a (distracting) virtual colormap. However, you cannot run XVSS if you do not have XView available (or cannot install it for some reason).

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, XVSS, or 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 `$LOAD`. However, if you are going to have, say, one *additional* display device (for a total of *two* alternates), you would *add another* set of lines like the above two, replacing the "`:0:`" with "`:2:`", and also `/YVTV/` and `$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. You should have the following line in the `$YPGVDEV` section of `LIBR.DAT`:

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

This will ensure that TVMON is built when *AIPS* is compiled. Note that this line does not depend on which of the servers is used; all three servers use the same Y and Z routines.

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 machines, 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 no matter which server you are running.

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. (If you are building XVSS, you will also want to copy the file `$YSERV/XVSS.UU` to `xvss.uu` into 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 `sh 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.] On non-UNIX systems, you will also have to copy the program `UNSHR.FOR` from the `$YSERV` area, compile it, then run it to unpack the `.SHR` file.

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) `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) `DESTDIR` is the destination directory for the SSS, XAS or XVSS binaries (this should be the same as the area $LOAD if you have installed *AIPS* on the display machine). At the end of this document, an alternative arrangement of the servers and clients is described. In that alternative, $SYSLOCAL is used.

(3) For the 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).

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 the XAS and XVSS servers only, `INCDIRS` 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 all three 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).

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	4	4
Images per plane	256	256	256
Size of planes	1142, 802 *	1142, 800 *	1140, 772 *
Max. grey scale intensity	111	199 *	111 *
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

where * indicates a system-dependent or controllable parameter. All of the dimensions quoted assume an 8-bit (often called "10-bit") SUN workstation. The Y dimensions given also assume a `TEXT_SPACE` parameter of 69 (XAS), 69 (XVSS, no shared memory), 48 (XVSS with shared memory), and 74 (XSSS).

This completes the installation.

SPECIAL FEATURES OF XVSS

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

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.