VMS/IRAF Installation and Site Manager's Guide

Doug Tody, Suzanne Jacoby
IRAF Group

Nigel Sharp
CCS

National Optical Astronomy Observatories†
June, 1993

ABSTRACT

This document describes how to install or update the VMS version of the portable IRAF system (Image Reduction and Analysis Facility). Installation instructions as well as procedures for improving performance, minimizing disk and memory usage, configuring the system for the local site, and interfacing local devices are given.

July 1, 1993

†Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

Contents

VMS/IRAF Installation and Site Manager's Guide

Doug Tody, Suzanne Jacoby
IRAF Group

Nigel Sharp
CCS

National Optical Astronomy Observatories†
June, 1993

1. Introduction

Before installing VMS/IRAF, one must 1) obtain the VMS/IRAF distribution, e.g., from the IRAF network archive on `iraf.noao.edu` (or by ordering a tape distribution from NOAO), 2) select the server or node on which the system is to be installed and arrange for sufficient disk space to hold the system, and 3) set aside sufficient time to do the installation. If these directions are followed carefully and mistakes are avoided the basic installation should only take an hour or so. Additional time may be required to customize the system to configure the local tape drives and other devices, set up IRAF networking, and so on.

V2.10 VMS/IRAF supports VMS versions 5.2 through 5.5. The amount of disk space required to install IRAF, including both the core system and NOAO package sources and all binaries, is about 78 Mb. If desired about half of this space can be reclaimed after installation by stripping the system of all but the runtime files.

```
$ dir/size=all irafdisk:[iraf...]
Grand total of 401 directories, 10507 files, 144566/157008 blocks.
```

The distributed system is a snapshot of our local (NOAO/Tucson) VMS/IRAF installation. The device and system configuration tables come edited for our system, and will have to be modified as part of the installation process. These modifications are discussed in detail in this document. To simplify the installation process as well as future upgrades, we have tried to isolate the site dependent files to the minimum number of directories, i.e., `dev`, `hlib`, and `local` and its subdirectories (the equivalent VMS pathnames are [IRAF.DEV], [IRAF.VMS.HLIB], and [IRAF.LOCAL]). The remainder of the system should not require any modifications.

Feel free to contact the IRAF Hotline for assistance if any problems should arise during the installation, or for help in interfacing new devices.

IRAF SUPPORT SERVICES

Internet:	iraf@noao.edu
NSI-DECNET:	noao::iraf or 5355::iraf
IRAF Hotline:	(602) 323-4160
FAX:	(602) 325-9360
Network Archive:	ftp iraf.noao.edu
(anonymous ftp)	ftp 140.252.1.1

†Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

2. Installing VMS/IRAF

The steps required to install VMS/IRAF are summarized below. This summary is provided only to introduce the installation procedure: it is important to read the detailed installation instructions for additional information, to avoid certain errors or omissions which are otherwise likely (IRAF does not always follow VMS conventions!).

```
# Prepare the root IRAF directory.
if new installation
    create iraf account
else if updating an old installation
    save locally modified files; delete old system

# Read in the distribution files.
login as iraf
unpack the distribution files from a network distribution or tape

# Modify IRAF system and device configuration files.
edit site dependent files in hlib, hlib/libc and dev directories
if updating an old installation
    merge locally modified files back into new system

# Complete the Initial Installation as SYSTEM.
add global symbol "iraf" to SYLOGIN.COM
copy edited hlib$libc/iraf.h file to SYS$LIBRARY
install in system memory IRAF shared library and other executables

# Checkout the new system.
run test procedure from iraf account
```

If problems should arise during the installation, help is available from the IRAF Hotline (602-323-4160), or by sending email to `iraf@noao.edu.`

2.1. Prepare the root IRAF directory

2.1.1. If updating an existing IRAF installation...

If you are updating an existing IRAF installation, you will be replacing IRAF by the new version and IRAF should already have an account and root directory on the desired host system. You should first log in as IRAF and save any locally modified files before deleting the old system. Ordinarily the only directories containing files you may wish to save are `dev`, `hlib`, and `local`. The safest and easiest way to do this is to save the entire contents of those directories. For example:

```
$ set default [iraf]
$ rename dev.dir,local.dir,[.vms]hlib.dir [directory]
```

This would physically move (not copy) the `dev`, `local`, and `hlib` directories to the named directory, which must be on the same disk device as [IRAF]. Many variations on this are possible. The destination directory should be somewhere *outside* the [IRAF...] directory tree, else it may get deleted when you delete the old system.

Although it is probably simplest and safest to save entire directories as in the example, in practice only a few files are likely to have been modified. These are the following:

```
[iraf.dev]tapecap.
[iraf.dev]graphcap.
[iraf.dev]termcap.
[iraf.dev]hosts.
[iraf.dev]uhosts.
[iraf.dev]irafhosts.
[iraf.dev]devices.
[iraf.dev]devices.hlp
```

```
[iraf.vms.hlib]motd.
[iraf.vms.hlib]login.cl
[iraf.vms.hlib]extern.pkg
[iraf.vms.hlib]install.lst
[iraf.vms.hlib]irafuser.com
[iraf.vms.hlib]zzsetenv.def
[iraf.vms.hlib.libc]iraf.h
[iraf.local]login.com
```

Having temporarily preserved all the locally modified files, it is now time to delete the old system and read in the new one. If you are the cautious (prudent?) type you may wish to first preserve the entire existing IRAF system on a backup tape. Using only the standard VMS utilities, the old system may be deleted as follows (assuming IRAF owns all the files in [IRAF...]).

```
$ set default [iraf]
$ delete [...]*.*;* /nolog/noconfirm
$ set protection=(owner:wd) [...]*.*;*
$ delete [...]*.*;* /nolog/noconfirm
    (repeat delete command with <ctrl/b> until done)
```

It is normal for the *delete* command shown to generate lots of messages during execution warning that it cannot delete directories because they are not yet empty. When the command finally executes without any warning messages this means the directory tree has been fully deleted.

Once the old system has been deleted you are ready to install the new one, as described in §2.2. It is important to delete the old system first to avoid creating junk files or directories when the new system is installed (due to file or directory name changes or deletions). Once the new system has been restored to disk, do *not* merely restore the files saved above, as you will need to carefully merge local changes into the versions of the files supplied with the new IRAF release.

2.1.2. If installing IRAF for the first time...

If you are installing IRAF for the first time then the first step is to set up a new account for the user ‘iraf’. This is necessary for IRAF system management, which should always be done from the IRAF account. Having an IRAF account provides a convenient place (the IRAF system manager’s login directory) to keep scratch files created during system configuration.

System management policies vary from site to site, so we do not give specific instructions for *how* to create the account (e.g., `$ run sys$system:authorize`), but the account itself should be structured as follows:

- Select a disk device with sufficient free space for the system (about 75 Mb for V2.10). If necessary, the system can be stripped after installation to save space (§8.1), but the full amount of space will be needed during installation. The disk should not be a "rooted logical" †.
- The root directory for the IRAF account should be set to [IRAF], as this name is explicitly assumed in various places in the configuration files. A "rooted logical" directory will not work.
- The *login* directory for the IRAF account should be set to [IRAF.LOCAL] rather than [IRAF] as one would expect, as we want to keep all the locally modified files in subdirectories off the iraf root directory, to simplify site management and future updates. If this point is missed the iraf environment will not be set up properly, and later problems are sure to result.

---

†A rooted logical might be something like `local_disk = dub4:[local.]`, with the IRAF disk set to `local_disk`. This will not work.

- Do not create a LOGIN.COM for IRAF; one will be read from the distribution tape, as will the [IRAF.LOCAL] directory. (If for some local management reason the directory [IRAF.LOCAL] is created at this time, make sure it has the proper protections [e.g., "`set protection = (s:rwd,o:rwd,g:r,w:r)`"].)
- There is no need to worry about VMS quotas and privileges yet; this is not a concern during installation and is discussed in a later section (§7.4).

2.2. Read in the distribution files

If you are installing from tape skip forward to §2.2.2. If you are installing from a network distribution (i.e., from disk) continue with the next section.

2.2.1. Installing from a network distribution

VMS/IRAF is available over the network via anonymous ftp from the node `iraf.noao.edu`, in the subdirectory `iraf/v`*nnn*`/VMS5`, where "`v`*nnn*" is the IRAF version number, e.g., subdirectory iraf/v210/VMS5 for V2.10 VMS/IRAF. Complete instructions for retrieving IRAF from the network archive are given in the README file in this directory. The retrieval can be accomplished with UNIX FTP, VMS FTP or VMS DECNET.

The subdirectory ia.vms5.vax contains the network version of the distribution file IA.VMS5.VAX. This single file contains the entire VMS/IRAF distribution, including both sources and binaries. The file IA.VMS5.VAX is stored in this directory as a compressed VMS backup save file, split into 512 Kb segments. To reconstruct the original backup save file, the segments must be transferred, uncompressed and concatenated. This can be done in several ways, as described in the README file. Once the disk save set has been reconstructed on your local disk, you can proceed with the installation.

Login as IRAF (ignore any LOGIN.COM not found error message) and run the VMS *backup* utility to read the backup save set, named v210.bck in this example:

```
$ set default disk:[iraf]
$ backup v210.bck/select=[iraf...] [...]/own=iraf/prot=w:r
```

In this and all the following examples, names like *disk:*, etc., denote site dependent device names for which you must supply values.

2.2.2. Installing from tape

Login as IRAF (ignore any LOGIN.COM not found error message) and run the VMS *backup* utility to read the backup tape or tapes provided.

```
$ mount/foreign tape-device:
$ set default disk:[iraf]
$ backup/rew tape:v210.bck/select=[iraf...] [...]/own=iraf/prot=w:r
```

The tape may be restored while logged in as SYSTEM if desired, but the switch /OWNER=IRAF should be appended to give the IRAF system manager (anyone who logs in as IRAF) write permission on the files.

It typically takes twenty minutes to half an hour to read the tape on a lightly loaded system.

2.2.3. Merge local revisions back into the new system

If this is a new IRAF installation this step can be skipped. Otherwise, once the new system has been restored to disk any local revisions made to the previous IRAF installation should be merged back into the new system. See §2.1.1 for a list of the files most likely to be affected. When propagating revisions made to these files, be sure to not replace the entire file with your saved version, as the version of the file in the new release of IRAF will often contain important additions or changes which must be preserved. It is best to merge your revisions into the version of the file which comes with the new IRAF.

This task will be easier if the revisions have been localized as far as possible, e.g., keep all `termcap` additions together at the head of the file, so that they may merely be transferred to the new file with the editor. The task of propagating revisions will also be much easier if detailed notes have been kept of all revisions made since the the last release was installed.

Beginning with IRAF version 2.8, one should no longer install locally added software in the core system LOCAL package. This significantly complicates updates and is no longer necessary as, due to the layered software enhancements introduced in V2.8 IRAF, it is now straightforward for each site to maintain their own custom LOCAL package external to the core IRAF system. The core system LOCAL is now only a *template-local* to be copied and used as the starting point for a custom LOCAL layered package. The layered software enhancements, and the procedure for building a custom LOCAL, are discussed in §9 of this manual.

2.3. Edit the system configuration files

The files listed below must be edited before the system can be run. The principal change required is to change the pathnames of the major IRAF logical directories for the local machine. If any of these files are edited while logged in as SYSTEM, be sure to do a `set file/owner=iraf` to restore ownership of the edited file to IRAF.

2.3.1. [IRAF.VMS.HLIB]IRAFUSER.COM

This file defines the VMS logical names and symbols needed to run IRAF. The site dependent ones are grouped at the beginning of the file. The top of the distributed file looks like this:

```
$! IRAFUSER.COM -- VMS symbols and logicals for using IRAF.
$!
$! The following logical names are system-dependent.  IMDIRDISK and
$! TEMPDISK should be public devices and may be the same device.
$!
$  define/job IRAFDISK   USR$3:    ! Disk IRAF resides on
$  define/job IMDIRDISK  SCR$4:    ! User image subdirectories go here
$  define/job TEMPDISK   SCR$2:    ! Runtime (scratch) files go here
$  define/job IRAFGMTOFF -7        ! Offset from GMT in hours
$! define/job IRAFUHNT   (file)    ! Use (file) instead of dev$uhosts
$!
```

`IRAFDISK`

Set to the name of the disk the [IRAF] directory is on; this may be another logical name but not a "rooted logical".

`IMDIRDISK`

Set to the name of a public scratch device, if available. The *mkiraf* script will try to create the default user image storage directories on this disk, e.g., IMDIRDISK:[*username*]. All potential IRAF users should have write permission and quota on this device.

A public scratch device is desirable because this is where bulk image data will be stored by default. It is often best for this to be on a different disk than that used for user logins, to minimize the amount of disk that has to be backed up on tape, and to permit a different quota and file expiration date policy to be used for large datafiles than is used for the small, relatively long lived files normally kept on the user disk.

Note for sites that use "rooted logicals": the VMS/IRAF kernel does not support rooted logicals. If you use one, e.g., for IMDIRDISK, batch jobs that need to access it will fail. To avoid problems, you must either choose IMDIRDISK to be truly just a disk specification, or inform all users to edit their IRAF login.cl files after a *mkiraf*.

For example, if you define IMDIRDISK to be "DATA_DISK:", where DATA_DISK is a logical name for dub4:[data.], user ICHABOD's login.cl would have "set imdir = DATA_DISK:[ICHABOD]", but this would not work for batch jobs. The user should edit login.cl after a *mkiraf* to use an expanded directory specification, in this case "set imdir = dub4:[data.ichabod]".

`TEMPDISK`

Set to the name of a public scratch device on which you have created a public directory named [IRAFTMP]. The device may be the same as is used for IMDIRDISK if desired. The IRAF logical directory "tmp" (known as IRAFTMP in the IRAFUSER.COM file) is defined as TEMPDISK:[IRAFTMP].† The IRAF system will create temporary files in this directory at runtime. These files are always deleted immediately after use (unless a task aborts), hence any files in IRAFTMP older than a day or so are garbage and should be deleted. It is best if IRAFTMP points to a public directory which is cleaned periodically by the system, e.g., whenever the system is booted, so that junk temporary files do not accumulate. This is a good reason for using a single public directory for this purpose rather than per-user private directories. The files created in IRAFTMP are rarely very large.

See note under IMDIRDISK concerning rooted logicals, if you plan to locate TEMPDISK at a rooted logical.

`IRAFGMTOFF`

The value of IRAFGMTOFF is the offset in hours to convert Greenwich mean time (GMT) to local standard time. It is used by the VMS/IRAF versions of *rtar* and *wtar* to restore file dates properly when moving files between UNIX and VMS systems. For example, Tucson is 7 hours west of Greenwich so IRAFGMTOFF is defined as -7.

`IRAFUHNT`

Define IRAFUHNT to be the name of the internet host table for the local system, if this exists and is in the correct format for use by IRAF (i.e., the same format as the file dev$uhosts). On VMS systems there is no standard internet host table comparable to the /etc/hosts file used on unix systems. The VMS/IRAF implementation of TCP/IP networking therefore uses its own host table, dev$uhosts, which is what is used if IRAFUHNT is not defined. Some VMS networking implementations (e.g., Multinet) have their own host tables. If this is the case then IRAF can make use of the system host table by uncommenting and defining the logical name IRAFUHNT to point to the file.

`FAST,BATCH,SLOW`

These are the logical names of the standard IRAF logical batch queues. They should be redefined to reference the queues used on your machine, e.g., the standard VMS batch queue SYS$BATCH (all three names may point to the same batch queue if desired). The fast queue is intended for small jobs to be executed at interactive priorities, the batch queue is for medium sized jobs, and the slow queue is for large jobs that need a lot of system resources and are expected to run a long time.

2.3.2. [IRAF.VMS.HLIB]INSTALL.COM

This command procedure is run by the system manager, or by the system at boot time, to install the IRAF shared library and selected executable images.†† Installing images in system

†If local system management or accounting policies require that private tmp and imdir directories be defined for each user, rather than the public ones we recommend, see Appendix A.

††In VMS terminology, *image*, as in "executable image" refers to the executable program, and has nothing to do with IRAF image data.

memory is necessary to permit memory sharing in VMS, and can greatly reduce physical memory usage and paging on a busy system, while speeding process startup. Installing images also consumes system global pages, however, of which there is a limited supply. Hence, one should only install those images which will be used enough to make it worthwhile. See §8.2 for more information.

The `install.com` procedure both installs the IRAF executables and replaces them after any have been changed. It works from a separate list of files, so you should not edit install.com itself. Instead, edit install.lst to select files by commenting out only those which are *not* to be installed (the comment character is a "!" at the beginning of the line).

The IRAF shared library `s_iraf.exe` should always be installed if IRAF is going to be used at all, or the system will execute very inefficiently (the IRAF shared library is used by all executing IRAF processes). Normally the executables cl.exe and x_system.exe should also be installed, since these are used by anyone using IRAF, as well as by all batch IRAF jobs. If IRAF is heavily used and sufficient global pages are available it is also desirable to install x_images.exe and x_plot.exe, since virtually everyone uses these executable images frequently when using IRAF. It is probably not worthwhile to install any other executables, unless usage at the local site involves heavy use of specific additional executable images.

2.3.3. [IRAF.VMS.HLIB.LIBC]IRAF.H

This file (often referred to as "<iraf.h>") is required to compile any of the C source files used in IRAF. Most sites will not need to recompile the C sources and indeed may not even have the DEC C compiler, but the file is also used by the runtime system in some cases to resolve logical names, hence must be edited and installed in the VMS system library. Change the following disk names as required for your system, referencing only system wide logical names in the new directory pathnames (e.g. usr1:[IRAF.VMS], where usr1 is a system logical name; do not leave the IRAF logicals like irafdisk in the definition).

IRAF Logical Directory	*VMS directory*	
HOST	*irafdisk*:[IRAF.VMS]	
IRAF	*irafdisk*:[IRAF]	
TMP	*tempdisk*:[IRAFTMP]	(may vary††)

These directory definitions are referenced only if logical name translation fails, as sometimes happens on VMS systems for various reasons. It is therefore essential that only system wide logical names be used in these directory pathnames. Do not use job or process logicals. Do not change the order in which the entries appear, or otherwise alter the syntax; the kernel code which scans <iraf.h> is very strict about the syntax.

2.4. Complete the initial system configuration

Login again as SYSTEM and copy the recently edited file irafdisk:[iraf.vms.hlib.libc]iraf.h to the system library, ensuring that the file has world read permission (be sure not to copy [iraf.vms.hlib]iraf.h by mistake):†

††If local system restrictions forbid a public IRAFTMP directory, set this entry to the pathname of a suitable directory in IRAF, e.g., [IRAF.LOCAL.UPARM]. This should work in most circumstances, since it is most likely to be the IRAF system manager who runs a program that accesses these pathnames. This is separate from the user-level private IRAFTMP irectory discussed in Appendix A.

†On a VMS cluster, make sure the IRAF.H file is added to SYS$COMMON:[SYSLIB] rather than SYS$SYSROOT:[SYSLIB], so that all nodes in the cluster may transparently access the file. The same precaution is needed when editing the system-wide login procedure file (e.g., SYLOGIN.COM), which will probably want to go into SYS$COMMON:[SYSMGR]. The SYSTARTUP_V5.COM file, on the other hand, should go into SYS$SYSROOT:[SYSMGR] if the bootstrap procedure is different for each node.

```
$ set default sys$library
$ copy disk:[iraf.vms.hlib.libc]iraf.h []/protection=w:r
```

Add the following statement to the system SYLOGIN.COM file, making the appropriate substitution for *disk*.

```
$ iraf :== "@disk:[iraf.vms.hlib]irafuser.com"
```

Add the following statement to the SYSTARTUP_V5.COM file, read by the system at startup time. This is necessary to cause the IRAF executables to be reinstalled in system memory when the system is rebooted.

```
$ @disk:[iraf.vms.hlib]install.com
```

While still logged in as SYSTEM, type in the above command interactively to perform the initial executable image installation. It may be necessary to increase the number of system global pages for the command to execute successfully. If you do not want to install the set of IRAF executables normally installed at NOAO, edit [iraf.vms.hlib]install.lst before executing install.com (see §8.2).

Lastly, log out of SYSTEM, and back in as IRAF. Update the date stamp of a file that is used to ensure that newly updated parameter sets are used rather than any old versions a user might have around:

```
$ set default irafhlib
$ copy utime. utime.
$ purge utime.
```

3. Testing a Newly Installed or Updated System

At this point it should be possible for any user to run IRAF. First however, you should verify this by using the IRAF account itself. Before testing a newly installed or upgraded system it is wise to read the *CL User's Guide*, the revisions notes, and the list of known bugs, if one has not already done so.

3.1. Configuring the IRAF account

The default LOGIN.COM in the IRAF login directory [IRAF.LOCAL] will automatically execute the globally defined system command 'iraf', which defines the IRAF logicals. Login as IRAF so IRAFUSER.COM is executed and the IRAF logicals are defined.

Before starting the IRAF CL (command language) you should do what a regular user would do, i.e., run *mkiraf* to initialize the IRAF uparm directory and create a new LOGIN.CL (answer yes to the purge query, as it is advisable to delete the contents of the old uparm). You may edit the LOGIN.CL created by *mkiraf* as desired.

```
$ set default [iraf.local]
$ mkiraf
```

Any modifications you wish to preserve across another *mkiraf* may be placed into an optional "loginuser.cl" file to avoid having to edit the login.cl file. The loginuser.cl file can have the same kinds of commands in it as login.cl, but *must* have the statement `keep` as the last line.

Once the IRAF environment is configured one need only enter the command *cl* to start up the CL. From an X windows workstation, you should always start the CL from an *xterm* window (or *xgterm* if available). See §6.1.

```
$ cl
```

If the CL runs successfully, the screen will be cleared (if the terminal type is set correctly) and the message of the day printed. You can then type *logout* to stop the CL and return to DCL, or stay in the CL to edit and test the device files described in the next section. When logging back into the CL (as ‘IRAF’), always return to the [IRAF.LOCAL] directory before logging in. A little command *h* (for ‘home’) is defined in the default IRAF LOGIN.COM file for this purpose.

If the CL does not run successfully, it is probably because the *iraf* command was not executed in the LOGIN.COM file at login time. The *iraf* command must be executed to define *cl* and the other VMS logical names and symbols used by IRAF, before trying to run the IRAF system.

Important Note: any users wishing to execute IRAF tasks in VMS batch queues *must* issue the ‘iraf’ command inside their LOGIN.COM, prior to any

```
if f$mode().nes."INTERACTIVE" then exit
```

command, otherwise critical IRAF logical names will be undefined when executing tasks remotely using IRAF networking.

Once in the CL, you should be able to draw vector graphics on the workstation screen (*xterm*) or graphics terminal and have printer access (device *vmsprint*). If working from a workstation on which you’ve started *saoimage* (see §6.2), you should be able to display images. In addition, you may have magtape access with minimum modifications to dev$tapecap and may or may not have graphics plotter access. If the graphics terminal capability is ready the next step is to run the IRAF test procedure to verify that all is working well, as well as try out a few of the many tasks in the system. If the graphics terminal is not up yet, there is probably no point in running the test procedure. To run the test procedure, read the documentation in the *IRAF User Handbook*, Volumn 1A, and follow the instructions therein. The test procedure is also available separately from the anonftp archive on iraf.noao.edu.

4. Configuring the IRAF Environment

Since IRAF is a machine and operating system independent, distributed system capable of executing on a heterogeneous network, it has its own environment facility separate from that of the host system. Host system environment variables may be accessed as if they are part of the IRAF environment (which is sometimes useful but which can also be dangerous), but if the same variable is defined in the IRAF environment it is the IRAF variable which will be used. The IRAF environment definitions, as defined at CL startup time, are defined in a number of files in the [IRAF.VMS.HLIB] directory. Chief among these is the system wide `zzsetenv.def` file. Additional user modifiable definitions may be given in the user’s login.cl file.

4.1. [IRAF.VMS.HLIB]ZZSETENV.DEF

The zzsetenv.def file contains a number of environment definitions. Many of these define IRAF logical directories and should be left alone. Only those definitions in the header area of the file should need to be edited to customize the file for a site. It is the default editor, default device, etc., definitions in this file which are most likely to require modification for a site.

If the name of a default device is modified, the named device must also have an entry in the dev$termcap file (terminals and printers) or the dev$graphcap file (graphics terminals and image displays). There must also be an *editor*.ed file in dev for each supported editor; *edt*, *emacs*, and *vi* are examples of currently supported editors.

Sample values of those variables most likely to require modification for a site are shown below.

```
set editor      = "vi"
set printer     = "versatec"
set stdplot     = "lw5"
set stdimage    = "imt512"
```

For example, you may wish to change the default editor to "emacs", the default printer to "vmsprint", or the default image display to "imt800". Note that the values of terminal and stdgraph, which also appear in the zzsetenv.def file, have little meaning except for debugging processes run standalone, as the values of the environment variables are reset automatically by the IRAF *stty* task at login time. The default stty setting in VMS/IRAF V2.10 is "xterm". This is consistent with the default values of terminal and stdgraph defined in zzsetenv.def to be "xterm" and would be appropriate for users with workstations running X based DECwindows/VMS or Motif/VMS. The issues of interfacing new graphics and image display devices are discussed further in §7.5.

4.2. The template LOGIN.CL

The template login.cl file, hlib$login.cl, is the file used by *mkiraf* to produce the user login.cl file. The user login.cl file, after having possibly been edited by the user, is read and executed by the CL every time a new CL is started to define the initial configuration of the CL. Hence this file plays an important role in establishing the IRAF environment seen by the user.

Examples of things one might want to change in the template login.cl are the commented out environment definitions, the commented out CL parameter assignments, the foreign task definitions making up the default USER package, and the list of packages to be loaded at startup time. For example, if there are host tasks or local packages which should be part of the default IRAF operating environment at your site, the template login.cl is the place to make the necessary changes.

5. Configuring the Device and Networking Files

The files listed below must be edited before IRAF can be used with the video terminals, graphics terminals, plotters, printers, magtape devices, and so on in use at the local site.

5.1. [IRAF.DEV]TAPECAP

Beginning with V2.10, IRAF magtape devices are described by the **tapecap** file, dev$tapecap. This replaces the old "devices" file for all runtime access to tape devices (the "devices" file is still used however in a lesser role; see below). The tapecap file describes each local magtape device and controls all i/o to the device, as well as device allocation.

The tapecap file included in the distributed system includes some generic device entries such as "9t-6250" (9track 1/2 inch tape drive at 6250 bpi), "vms-exabyte" (exabyte EXB-8200 8mm tape drive on VMS), and so on which you may be able to use as-is to access your local magtape devices. For example, by default, the device known to the CL as "mta" is a TA78 9tk drive known as device TU0:, which is locally defined at the system level as a specific tape drive (actually, in our cluster, $1$MUA0:). Notice how related entries are chained together in the tapecap file with the tc= field:

```
mta|mtahi|mta6250|TA78 9tk tape drive|         :tc=mttu0-6250:

mttu0-6250|mttu0|TA78 half inch reel tape drive (VMS):\
                 :al=tu0:dv=tu0:lk=tu0:mr#64512:or#64512:se:tc=9tk-6250:

9tk-6250|9track 1/2 inch tape drive at 6250 bpi:\
                 :dt=9 track at 6250 bpi:tt=2400 foot:ts#140000:\
                 :dn#6250:bs#0:mr#65535:or#65535:fb#10:
```

Most likely you will want to add some device aliases, and you may need to prepare custom device entries for local devices. There must be an entry in the tapecap file for a magtape device in order to be able to access the device from within IRAF.

Instructions for adding devices to the tapecap file are given in the *IRAF V2.10 Revisions Summary*. Notes on the VMS specific implementation of the new magtape facilities are found in §7.2 of this document.

5.2. [IRAF.DEV]DEVICES

This file is used to associate IRAF logical device names with VMS device names. This file is no longer used by the IRAF magtape i/o system which is now all tapecap based, but is used by host level utilities like *devstatus* which need a simple way to translate IRAF device names to host device names. The file is optional and need not be configured to run IRAF.

5.3. The DEVICES.HLP file

All physical devices that the user might need to access by name should be documented in the file dev$devices.hlp. Typing

```
    cl> help devices
```

or just

```
    cl> devices
```

in the CL will format and output the contents of this file. It is the IRAF name of the device, as given in files such as termcap, graphcap, and tapecap, which should appear in this help file.

5.4. [IRAF.DEV]TERMCAP

There must be entries in this file for all video terminal, graphics terminal, and printer devices you wish to access from IRAF. The printer is easy, since the "vmsprint" entry provided should work on any VMS system. To prepare entries for other devices, simply copy the vmsprint entry and change the queue name from SYS$PRINT to the name of the queue for the new printer.† Any number of these entries may be placed in the termcap file, and there can be multiple entries or aliases for each device. If you have a new terminal which has no entry in the termcap file provided, a new entry will have to be added. Termcap is widely used, so it is highly likely that someone somewhere will already have written a termcap entry for your terminal. If the terminal in question is a graphics terminal with a device entry in the graphcap file, you should add a ‘:gd’ capability to the termcap entry. If the graphcap entry has a different name from the termcap entry, make it ‘:gd=*gname*’. Local additions should be placed at the top of the file to make it easier to merge the changes into a future IRAF release.

5.5. [IRAF.DEV]GRAPHCAP

There must be entries in the graphcap file for all graphics terminals, batch plotters, and image displays accessed by IRAF programs. New graphics terminals will need a new entry. The IRAF file gio$doc/gio.hlp contains documentation describing how to prepare graphcap device entries. A printed copy of this [60 page] document is available from the iraf/docs directory in the IRAF network archive. However, once IRAF is up you may find it easier to generate your own copy using *help*, as follows:

```
    cl> help gio$doc/gio.hlp fi+ | lprint
```

which will print the document on the default IRAF printer device (use the "device=" hidden parameter to specify a different device). Alternatively, to view the file on the terminal,

```
    cl> phelp gio$doc/gio.hlp fi+
```

The help pages for the IRAF tasks *showcap* and *stty* should also be reviewed as these utilities are useful for generating new graphcap entries. The i/o logging feature of *stty* is useful for determining exactly what characters your graphcap device entry is generating. The *gdevices* task is useful for printing summary information about the available graphics devices.

†If the printer or plotter device is on a remote node which is not clustered with the local node but which is accessible remotely, IRAF networking must be used to access the remote device. IRAF networking is also frequently used to access devices on non-VMS (e.g., UNIX) nodes.

Help preparing new graphcap device entries is available if needed. We ask that new graphcap entries be sent back to us so that we may include them in the master graphcap file for all to benefit.

5.6. [IRAF.DEV]HOSTS, UHOSTS, and IRAFHOSTS

The dev directory contains several files (`hosts`, `irafhosts`, and `uhosts`) used by the IRAF network interface. IRAF networking is used to access remote image displays, printers, magtape devices, files, images, etc., via the network. IRAF network nodes do not have to have the same architecture, or even run the same operating system, i.e., one can access a UNIX/IRAF node from VMS/IRAF and vice versa. V2.10 IRAF contains extensive changes to the networking code to permit remote client access to a VMS node via TCP/IP, and runtime selection of either TCP/IP or DECNET for the transport method. Editing changes to the networking related files in [IRAF.DEV] are illustrated here. Further documentation on the networking interface can be found in the *IRAF Version 2.10 Revisions Notes* and §7.1 of this manual.

The `dev$hosts` file is used to configure IRAF networking on the local system. This file tells IRAF what network nodes are defined and how to connect to them, including the protocol to be used. Initially IRAF networking is disabled. To enable networking there must be an entry in the hosts file for the host IRAF is running on. To do anything useful with networking there must also be entries for the nodes you wish to connect to. Often a single VMS/IRAF installation and a single copy of dev$hosts are shared by several nodes in a VMS cluster.

To connect to a local VMS node via DECNET requires an entry in the hosts file like the following (these and the following examples are taken from the NOAO VMS/IRAF hosts file).

```
robur r                 : robur::0=irafks
draco                   : draco::0=irafks
```

Here, nodes "robur" and "draco" are defined as DECNET-connected nodes with the IRAF kernel server (IRAFKS) installed as a "zero numbered" object in DECNET. IRAFKS can also be installed as a numbered DECNET service, in which case the hosts file entry should appears as follows.

```
robur r                 : robur::242
draco                   : draco::242
```

In this example the IRAFKS service is installed locally as object number 242 in DECNET.

To connect to a node on a remote UNIX host via TCP/IP requires an entry using "!" to indicate the TCP/IP protocol, followed by the UNIX command to be executed to start up the IRAF kernel server, as in the following examples (note the different machine types).

```
cephus                  : cephus!/usr/iraf/bin.ddec/irafks.e
columba                 : columba!/usr1/iraf/iraf/bin.mips/irafks.e
felis                   : felis!/u1/iraf/iraf/bin.rs6000/irafks.e
lepus                   : lepus!/usr/iraf/bin.f2c/irafks.e
noctua                  : noctua!/usr/iraf/iraf/bin.irix/irafks.e
ursa u                  : ursa!/local/iraf/bin.sparc/irafks.e
vega                    : vega!/iraf/iraf/bin.hp700/irafks.e
```

Finally, the hosts file may contain logical node entries. These are not real network nodes, but hosts file entries which point to some other entry in the same host table. These are most commonly used to designate logical nodes for disposing of output to different classes of devices such as printers and plotters, hosted by some node elsewhere on the network, but can be used for other purposes such as gateways.

```
# Logical nodes (lpnode = line printer output, plnode = plotter output).
lpnode                  : @ursa
plnode                  : @ursa
```

In the example above node ursa is a Sun server; the hosts file entry for this node is given in an earlier example above.

The other files required to configure IRAF networking are the `uhosts` and `irafhosts` files. The `uhosts` file is a standard internet host table containing hostnames and internet numbers. It is used for TCP/IP networking unless the logical name IRAFUHNT is defined to point to an alternate file (e.g., Multinet uses a hosts file which IRAF networking can access directly). The uhosts file, if used, should be replaced by a host listing for your local site. One way to do this is to copy the /etc/hosts file from a local UNIX host. Note that V2.10 VMS/IRAF TCP/IP networking does not support runtime lookup of hostnames via a name service, hence an internet host must have an entry in the uhosts or IRAFUHNT file for VMS/IRAF to be able to connect to the host.

The `irafhosts` file supplies login information used by IRAF networking when a connection is made. When a network login is attempted the networking system will use the user's .irafhosts file if there is one, otherwise it defaults to the irafhosts file in dev. The system version of irafhosts should not need to be modified unless proxy logins are enabled on the local host. Proxy logins are used to eliminate password prompts when making network connections between cooperating machines in the local VMS cluster. This is a desirable but optional feature in all but one case, network gateways. When IRAF network connections are routed through a local VMS machine serving as a gateway machine password prompts are not possible, and one must either enable proxy logins or supply each user with a .irafhosts file containing their password. Proxy logins are probably the least objectionable of these two options.

Thus far we have discussed the IRAF networking configuration only as it appears from the VMS side of the network, i.e., what one sees when configuring the dev$hosts and other files on a VMS host. This is sufficient to allow one VMS/IRAF host to network to another, or to allow a VMS/IRAF host to connect to an external host machine running UNIX. The remaining case occurs when IRAF running on a UNIX host connects to VMS/IRAF. To enable such "incoming" TCP/IP network connections one must make an entry for each VMS host in the dev$hosts file on the UNIX/IRAF (client side) host, as in the following example.

```
# VMS (decnet) nodes, via internet gateway robur.
robur r                 : -rcb robur!irafks
draco                   : @robur!draco
grian                   : @robur!grian
lenore                  : @robur!lenore
vela                    : @robur!vela
```

In this example all incoming (TCP/IP) connections are routed via the VMS gateway machine robur. The "irafks" is a DCL symbol defined on the server node in hlib$irafuser.com which runs the IRAF kernel server, irafbin:irafks.exe. The "@robur!nodename" syntax for the VMS nodes other than robur tells IRAF running on the UNIX host to use node robur as the gateway. The connection to node robur is made via the rexec-callback IRAF networking connect protocol (see §7.1 and also the V2.10 IRAF system notes file). The connection from the gateway machine to the target VMS node is controlled by the dev$hosts file on the *gateway* (VMS) node, and will often be a DECNET connection, hence the gateway can serve to connect the TCP/IP and DECNET domains.

Once again, for gateway connects to succeed either proxy logins must be enabled on the gateway machine, or each user must have a .irafhosts file on the gateway machine containing password information. For incoming TCP/IP network connections to work the VMS node must support the *rexec* network service, which is only available via third party networking packages such as Multinet. However, since IRAF networking supports gateways only the gateway host need run this software. When properly configured, gateways are transparent to the user.

To fully configure IRAF networking some VMS-level configuration is necessary in addition to editing the hosts, uhosts, and irafhosts files (for example IRAFKS should be defined as a known DECNET service). This is discussed further in §7.1.

6. Vector Graphics and Image Display using X Windows

In this section we discuss some DECwindows/VMS and Motif/VMS (X window system) utilities for use with IRAF running on VMS workstations. Further information can be found in the file `0readme` in the [IRAF.VMS.X11] directory. Note that VMS/IRAF can also be used with an X user interface running on a remote UNIX workstation, with only VMS/IRAF running on the VMS host. This section is relevant only if you need to run the X clients directly on a VMS host. We discuss only *xterm* and *saoimage* here since these were the only X clients available for graphics and image display at the time this was written, but newer facilties may be available before this document is again revised. Check with the IRAF group for more current information on support for graphics and imaging under the X window system.

6.1. Xterm

Xterm is a public-domain version of the standard MIT X window system terminal emulator, modified to work on VMS. It is a two-window tool capable of emulating a VT102 terminal and a Tektronix 4014 terminal. Appropriate escape sequences will cause *xterm* to switch between these two modes, but some programs require the user to make the change (using the CTRL-middle mouse button menu). It is especially valuable for IRAF use since it allows a plotted graph to be kept in the Tektronix emulation window while commands are given in the VT102 emulation window.

Due to significant changes between VMS and DECwindows versions, there are three different versions of *xterm*:

1) Xterm 1.1 (XTERM11.EXE) - for VMS 5.1 to 5.3-x, with DECwindows
2) Xterm 1.2 (XTERM12.EXE) - for VMS 5.4-x and later, with DECwindows
3) Xterm 2.1 (XTERM21.EXE) - for VMS 5.4-x and later, with DECwindows/Motif

A common mistake is to run the CL from a DECterm window, which does not support IRAF graphics. To use IRAF interactive graphics tasks one must run IRAF from an *xterm* window (or any similar terminal emulator supported by IRAF, e.g., *gterm*, *xgterm*, etc.). To use *xterm* with IRAF we recommend that the "xtermr" graphcap entry be used. This is set as follows:

```
    cl> stty xtermr nl=54
```

By default, this entry has 65 lines in the text window. If your setup is different (perhaps 54 lines instead of 65) you can make changes as shown on the stty command line. The "xterm" stty setting is an alias for the "xtermr" stty setting; they can be used interchangeably.

The xtermr graphcap entry writes the status line in the TEK window, not in the text window. This is the preferred setup, since DEC hasn't yet made available the customize feature for "focus follows cursor", meaning there is no way around needing to click-to-focus. Use of xtermr requires the least amount of clicking.

You should not click inside the TEK window to focus on it, but on its header or border. Otherwise, the mouse click returns a character which terminates cursor mode. Clicking on the border provides focus without reporting any events, since the cursor is not inside the "active" region.

6.2. SAOimage

Saoimage is an X-windows based image display server originally developed by the Smithsonian Astrophysical Observatory. It provides a large selection of options for zooming, panning, scaling, coloring, pixel readback, display blinking, and region specifications. User interactions are usually performed with the mouse. The version of *saoimage* included with the VMS/IRAF V2.10 distribution is 1.07; newer versions may be available from the IRAF archives. You must have *saoimage* running, although it can be iconified, before attempting to write to it. More information may be found in the `0readme` file in the [IRAF.VMS.X11] directory.

7. Additional System Configuration Issues

7.1. New networking driver

This section contains notes on the new V2.10 IRAF networking interface as implemented for VMS. Further information can be found in §5.6 of this manual, the *IRAF Version 2.10 Revisions Summary*, and the VMS/IRAF system notes file, iraf$local/notes.vms.

The VMS/IRAF networking code was extensively revised for V2.10 to permit client access to a VMS node via TCP/IP, and runtime selection of TCP/IP or DECNET for the transport method.

Selection of the transport protocol is made in the dev$hosts file, as in the examples below.

```
robur r          : robur::0=irafks
ursa u           : ursa!/local/iraf/bin.sparc/irafks.e
```

The first example above states that logical node robur, with alias r, is a DECNET domain node named robur. The DECNET connection is to be made by executing the command "irafks" on the remote node. User login information is supplied by the user’s .irafhosts file if any, otherwise by dev$irafhosts. A numbered DECNET object could be used instead, in which case the syntax on the right would change to "robur::*n*", where *n* is the number of the DECNET object to connect to.

The second example states that logical node ursa, with alias u, is a TCP/IP domain node with network name ursa. The network connection is made by executing, on the remote node ursa, the host command given after the "!". VMS/IRAF supports only the IRAF *rexec* connect protocol for outgoing connections to TCP/IP servers. TCP/IP connections are discussed in more detail below.

7.1.1. DECNET networking

In the simplest form of a DECNET connection the hosts file entry for the server node will reference a zero-numbered network object as in the example, e.g., "hostname::0=irafks" (or "hostname::task=irafks"). If "irafks" is *not* a defined DECNET network object then the user must have a file IRAFKS.COM in their login directory, similar to the following:

```
$! IRAFKS.COM -- Fire up the iraf kernel server under DECNET.
$!
$  if f$mode() .nes. "NETWORK" then exit
$!
$! iraf                          ! uncomment if not in login.com
$  irafks :== "$irafbin:irafks" ! define irafks as a foreign task
$  irafks dn.irafks             ! start the iraf kernel server
$  logout
```

This works, but has the disadvantage that every user account must have an IRAFKS.COM file in the login directory, or IRAF DECNET-based networking will not work. A better approach is to define IRAFKS as a known network object which will execute the file IRAFKS.COM supplied in the IRAFHLIB directory with V2.10 VMS/IRAF. This eliminates the need for each user to have a private copy of the IRAFKS.COM file, and makes proxy logins possible, which can eliminate the need for password prompts.

The following is an example of how to install irafks as a known network object. This must be done on each DECNET host. The device name USR$3: shown in the example should be replaced by the value of IRAFDISK: on the local system.

```
$ run sys$system:ncp
ncp> define object irafks -
    number 0 file usr$3:[iraf.vms.hlib]irafks.com
ncp> set object irafks -
    number 0 file usr$3:[iraf.vms.hlib]irafks.com
ncp> exit
```

In normal operation the NCP configuration is automatically preserved when the system is rebooted. If the local system has a network configuration procedure to redefine the NCP settings in the event of damage to the network database, then an NCP DEFINE command similar to that shown should be added to this file.

An alternative to the above scheme is to define IRAFKS as a numbered system object. This works as above, except that the "number 0" becomes "number *n*", where *n* is the network object number, and the dev$hosts file syntax for the server becomes "*hostname*::*n*".

7.1.2. Proxy logins

The DECNET *proxy login* capability allows DECNET connections to be made between a set of well-known cooperating systems (e.g., the local cluster) without the need to supply login authentication information every time a network connection is made. The effect is to eliminate password prompts, thereby making networking much more transparent to the user. In some cases eliminating password prompts is necessary to make the software function correctly, for example when using a VMS host as a gateway machine interactive prompting for passwords is not possible, and the gateway cannot function correctly without some way to disable password prompts. In the case of IRAF networking this can be done by having the user put password information in their .irafhosts file, but the use of proxy logins is preferred since it avoids plain-text passwords and avoids the need for the .irafhosts file altogether.

To enable proxy logins for IRAF networking one must first define IRAFKS (the IRAF kernel server) as a known network object, as outlined in the previous section. The VMS *authorize* utility is then used to enable proxy logins.

For example, assume we have two nodes ROBUR and DRACO.

```
$ run sys$system:authorize                  ! run on node robur
authorize> add/proxy draco::* */default

$ run sys$system:authorize                  ! run on node draco
authorize> add/proxy robur::* */default
```

This would enable proxy logins between nodes draco and robur for all user accounts that have the same user login name on both systems. Alternatively one could do this for only a specified set of user accounts. The authorize utility automatically updates the affected disk files so that the change will be preserved the next time the system is booted.

To eliminate the password prompts during IRAF networking connections one must also edit the user .irafhosts file, or the system irafhosts file (in dev) to disable login authentication. For example, if the dev$irafhosts file contains

```
# dev$irafhosts

draco       :  <user> none
robur       :  <user> none
*           :  <user> ?
```

then authentication will be disabled for connections made between nodes draco and robur, but a password will be required to connect to any other node. The above irafhosts file would result in a DECNET login request such as

```
hostname"username"::"0=irafks"
```

which forces a login as "username" on the remote host. If the user name is omitted, the default proxy login on the remote node is used. The user's .irafhosts file, if present, will be used instead and should be configured similarly.

These changes make IRAF DECNET based networking transparent between cooperating nodes in a local network, without requiring users to place password information in .irafhosts files. This is especially desirable if a routing node is used to route IRAF networking connections between TCP/IP and DECNET networks.

7.1.3. Configuring a VMS host as a TCP/IP server

For incoming TCP/IP connections (VMS node acting as server) VMS/IRAF supports two connect protocols, *rexec* and *rexec-callback*. The *rsh* protocol, the default connect protocol for V2.10 UNIX/IRAF systems, is not supported by VMS/IRAF. To act as a TCP/IP server a VMS node must run a host-level networking package (such as Multinet) which supports the *rexecd* networking daemon.

On the UNIX side the dev$hosts entry for a directly-connected VMS node should be similar to the following example.

```
robur r                 : -rcb robur!irafks
```

This specifies that for logical node robur, alias r, the rexec-callback connect protocol (TCP/IP) should be used to connect to the network node robur. The command to be executed on the remote node is whatever is to the right of the "!", i.e., the command "irafks" in our example.

When the *rexecd* daemon executing on the remote VMS node responds, the first thing it will do is login using the username and password supplied with the rexec request, and execute the user's LOGIN.COM file. For an IRAF user this must contain the command *iraf*, which causes the IRAFUSER.COM file in [IRAF.VMS.HLIB] to be interpreted by DCL. This defines all the IRAF logical names and symbols. One of the sybols defined is the *irafks* command which is subsequently executed to complete the rexec network request. The *irafks* command is defined in IRAFUSER.COM as follows:

```
$  irafks       :== $irafbin:irafks     !  IRAF network kernel server
```

Hence, as might be expected, the result of the rexec connect is to run the IRAF kernel server irafbin:irafks.exe on the server node. In the case of the rexec-callback connect protocol, the actual command passed to DCL includes command line arguments to the IRAF kernel server telling it how to call the client back on a private network socket. What happens is that the client reserves a port and creates a socket on this port, then issues a rexec call to the VMS system to run the kernel server. The port and host name are passed on the rexec command line. The *rexecd* daemon on the server runs the irafks.exe process on the VMS node, and irafks calls the client back on the socket reserved by the client. The end result is a high bandwidth direct socket connection between the client and server processes.

See the discussion in §5.6 for a discussion of TCP/IP and DECNET internetworking, including examples of how to configure a VMS node as an Internet gateway for IRAF networking.

7.1.4. Network security alarms

One potential consequence of IRAF networking is that, if IRAF networking is not properly configured (either at the system level or at the user level), network connection attempts may fail, and IRAF in its normal operation may repeatedly generate failed connection attempts. Repeated failed connection attempts can trigger network security alarms on a VMS system, even though no real security problem exists. VMS systems managers should be aware of this possibility so that they don't waste time trying to track down a potential network security problem which doesn't exist. Should this circumstance occur, the best solution is to properly configure IRAF networking so that the connection attempts succeed, e.g., by setting up the user's .irafhosts file, or by enabling proxy logins for the user.

7.2. New magtape driver

This section contains notes on the new V2.10 IRAF magtape driver as implemented for VMS. The reader is assumed to be familiar with the basics of magtape access in IRAF, including remote access to a drive via IRAF networking. Further information can be found in §5.1 of this manual, in the *IRAF Version 2.10 Revisions Summary*, and in the VMS/IRAF system notes file, iraf$local/notes.vms.

7.2.1. Allocation

Explicit allocation of magtape devices is now optional. If the drive is not allocated with the CL *allocate* command it will be automatically allocated and mounted when the first tape access occurs. Once allocated, it stays allocated even if you log out of the CL and back in, allowing successive IRAF or other processes to access the drive without it being deallocated.

The tape drive can be allocated at the VMS level (e.g., "allocate tu4:") before starting the CL. This reserves the drive but if you run *devstatus* in the CL it will report that the drive has not been "allocated", because IRAF doesn't consider the drive allocated in the IRAF sense unless it has been both allocated and mounted at the VMS level. One can either allocate the drive in the CL, or run an IRAF magtape task which accesses the drive to cause the drive to be automatically allocated and mounted.

If the drive is allocated in DCL before starting IRAF, then allocated and deallocated in the CL, when you log out of the CL the drive will still be allocated at the VMS level even though it was deallocated in the CL. If the drive is NOT allocated in DCL before starting the CL, allocating and deallocating the drive in the CL and then exiting will result in the drive not being allocated at the VMS level.

7.2.2. Density

When a drive is automounted it is possible to set the density. The V2.10 implementation supports this (you just write to mta1600 or whatever), but this does not work reliably since VMS requires that the first operation following a mount which changes the density be a write. Often the write is preceded by a file positioning operation such as a rewind or seek to end of tape, which prevents the density from being changed.

To reliably change the tape density, do an "init/density=NNN device" at the DCL level and verify the new density with "show magtape" ("!init", "!show" in IRAF). It may be necessary to repeat the operation before it succeeds.

7.2.3. Status output logging

Status output logging is fully implemented in VMS/IRAF, i.e., one can enable status output and direct the messages to a text file, a terminal, or a network socket.

For VMS/IRAF the magtape naming conventions have been extended slightly to allow the different types of status output logging devices to be indicated.

:so=.foo	log to file "foo"
:so=>foo	log to terminal "foo:"
:so=foo	log to socket on node "foo"

For example, "mtexamine mta[:so=orion]" would examine the files on drive `mta`, optionally sending status output messages to a tape status daemon running on network node orion. Logging to a named terminal device will usually fail due to permission problems, but logging to ">tt" will direct status output to the current terminal window.

The VMS version of the driver will print a device type string such as

```
    $2$MUA1: (ROBUR) - 9 track at 6250 bpi
```

where the first part of the message comes from VMS and the " - ..." is from tapecap. The density field can be either the tapecap value or the runtime GETDVI value. For example, if the density is shown as "6250" this is the value from the tapecap entry. If the density is shown as "(6250)", i.e., in parentheses, this is the actual device value as determined by a GETDVI call to VMS. Note that it is possible for the density given in the tapecap entry to differ from the actual device density setting. In this case the tape will be written correctly, but the "Device Type", "Capacity" and "Tape Used (%)" status fields will be incorrect. To avoid this problem be sure the density of the tapecap entry you are using to access the drive matches the density value of the physical device.

To enable status output logging to a socket a network magtape status display server such as *xtapemon* must be started on the remote node before accessing the tape drive. To permanently direct status output to a node a statement such as ’set tapecap = ":so=*hostname*"’ may be added to the login.cl or loginuser.cl file.

7.2.4. Using tape drives over the network

Changes in the VMS/IRAF V2.10 magtape and networking interfaces make it possible to remotely use VMS tape drives from Internet (UNIX) or DECNET hosts. In practice this is straightforward, but there are two caveats to keep in mind: 1) do not explicitly allocate the device, 2) if you run IRAF tasks in different packages that access the same tape drive, type *flpr* before running a task in a different package. Aside from these two items, everything should work as when accessing a magtape device directly on a UNIX or VMS system. Both problems arise from the way VMS handles device allocation.

Explicitly allocating a device does not work in VMS/IRAF when remotely accessing a tape drive because the effect is to allocate the drive to the kernel server process for the client CL, thereby preventing other client IRAF processes from accessing the drive. In practice this is not a problem since it is not necessary to explicitly allocate the device; the VMS/IRAF kernel server will automatically allocate and mount the device when it is accessed over the network.

An example of the problem of trying to access a remote VMS drive from two different IRAF packages occurs when the *rewind* task, which is in the SYSTEM package, is used in conjunction with DATAIO tasks such as *rfits*, *wfits*, or *mtexamine*. You can run tasks in a given package at will, but if you try to access the drive with a task in a different package there will be a conflict. Again, the problem is that when remotely accessing a device, the VMS device allocation facilities cause the device to be allocated to the kernel server *process*, rather than to the user. There is a one-to-one correspondence between client-side IRAF processes and kernel server processes. In IRAF each package has its own executable, and hence its own kernel server, leading to the allocate conflict.

For example if you have been using *rfits* (DATAIO) and you want to do a *rewind* (SYSTEM), type "flpr rfits" and wait 5-10 seconds, then you should be able to do the rewind. To run another DATAIO task, type "flpr rewind" and again wait 5-10 seconds before running the DATAIO task. The delay is necessary to wait for VMS to shutdown the kernel server for the package which previously had the device allocated.

7.3. Configuring user accounts for IRAF

User accounts should be loosely modeled after the IRAF account. All that is required for a user to run IRAF is that they run *mkiraf* in their desired IRAF login directory before starting up the CL. Each user should review the resulting login.cl file and make any changes they wish. Any user wanting to run batch jobs, including printer access, must execute the ‘*iraf*’ command in their LOGIN.COM file, making sure it is executed for non-interactive jobs. Individual process quotas for IRAF users should be set as described in the next section.

7.4. VMS quotas and privileges required to run IRAF

The only privilege required by IRAF is TMPMBX, which is probably already standard on your system. Systems with DECNET capabilities should also give their users NETMBX privilege, although it is not required to run IRAF. No other privileges are required or useful for normal activities.

Although privileges are no problem for VMS/IRAF, it is essential that the IRAF user have sufficient VMS quota, and that the system tuning parameters be set correctly, otherwise IRAF will not be able to function well or may not function at all. If a quota is exceeded, or if the system runs out of some limited resource, the affected VMS system service will return an error code to IRAF and the operation will fail (this frequently happens when trying to spawn a connected subprocess). The current recommended ranges of per-user quotas are summarized below. Users running DECwindows/Motif may already have, and may definitely need, larger numbers for these quotas, so don't reduce them to these values.

quota	*minimum*	*recommended*
BYTLM	20480	30720
PGFLQUOTA	15000	30720
PRCLM	5	10
WSDEFAULT	512	512
WSEXTENT	4096	WSMAX
JTQUOTA	2048	2048
FILLM	30	60

The significance of most of these quotas is no different for IRAF than for any other VMS program, hence we will not discuss them further here. The PRCLM quota is especially significant for IRAF since an IRAF job typically executes as several concurrent processes. The PRCLM quota determines the maximum number of subprocesses a root process (user) may have. Once the quota has been reached process spawns will fail causing the IRAF job or operation to abort.

The minimum number of subprocesses a CL process can have is 1 (x_system.e). As soon as a DCL command is executed via OS escape a DCL subprocess is spawned, and we have 2 subprocesses. The typical process cache limit is 3, one slot in the cache being used by x_system.e, hence with a full cache we have 4 subprocesses (the user can increase the process cache size if sufficient quota is available to avoid excessive process spawning when running complex scripts). It is common to have one graphics kernel connected, hence in normal use the typical maximum subprocess count is 5. However, it is conceivable to have up to 3 graphics kernel processes connected at any one time, and whenever a background job is submitted to run as a subprocess a whole new subprocess tree is created. Hence, it is possible to run IRAF with a PRCLM of 5, but occasional process spawn failures can be expected. Process spawn failures are possible even with a PRCLM of 10 if subprocess type batch jobs are used (the default), but in practice such failures are rare. If all batch jobs are run in batch queues it should be possible to work comfortably with a PRCLM of 5-6, but in practice users seem to prefer to avoid the use of batch queues, except for very large jobs.

Since IRAF uses memory efficiently the working set parameters do not seem critical to IRAF, provided the values are not set unrealistically low, and provided WSEXTENT is set large enough to permit automatic growth of a process working set when needed. Configuring VMS to steal pages from inactive processes is not recommended as it partially cancels the effect of the process cache, causing process pagein whenever a task is executed. It is better to allow at least a minimum size working set to each process. However, this is not a hard and fast rule, being dependent on individual system configurations and workloads.

In addition to sufficient per user authorized quota, the system tuning parameters must be set to provide enough dynamically allocatable global pages and global sections to handle the expected load. If these parameters are set too small, process connects will fail intermittently, usually when the system load is high. Each subprocess needs about 8 global pages when activated (IRAF uses global pages and shared memory for interprocess communications, due to the relatively low bandwidth achievable with the VMS mailbox facilities).

With IRAF in heavy use (i.e., a dozen simultaneous users) this can easily reach a requirement for several hundred additional global pages. Each installed image and subprocess also needs at least one, usually two, global sections. Note that the size of the executable found by doing a DIR/SIZE=ALL on [IRAF.BIN]*.EXE can be considered an upper bound to the number of pages needed to install it (if anyone wants to play it safe: typically, it's about 50-70 percent of this size). Currently, for 2.10, we have CL=357, S_IRAF=1267, IRAFKS=23, X_SYSTEM=106, X_PLOT=159, and X_IMAGES=786. The system parameters on our 8600 (which is probably a worst case example) are currently set to GBLPAGES = 12120 and GBLSECTIONS = 230. For every increment of 512 in GBLPAGES, GBLSECTIONS must be increased by 4. After making any of these changes, we recommend running AUTOGEN to ensure correct relationships among the many sysgen parameters.

7.5. Interfacing new graphics devices

There are three types of graphics devices that concern us here. These are the graphics terminals, graphics plotters, and image displays.

7.5.1. Graphics terminals

The IRAF system as distributed is capable of talking to just about any conventional graphics terminal or terminal emulator, using the stdgraph graphics kernel supplied with the system. All one need do to interface to a new graphics terminal is add new graphcap and termcap entries for the device. This can take anywhere from a few hours to a few days, depending on one's level of expertise, and the characteristics of the device. Be sure to check the contents of the dev$graphcap file to see if the terminal is already supported, before trying to write a new entry. Useful documentation for writing graphcap entries is the GIO reference manual and the HELP pages for the *showcap* and *stty* tasks (see §5.5). Assistance with interfacing new graphics terminals is available via the IRAF Hotline.

7.5.2. Graphics plotters

The current IRAF system comes with several graphics kernels used to drive graphics plotters. The standard plotter interface is the SGI graphics kernel, which is interfaced as the tasks *sgikern* and *stdplot* in the PLOT package. Further information on the SGI plotter interface is given in the paper *The IRAF Simple Graphics Interface*, a copy of which is available from the network archive.

SGI device interfaces for most plotter devices already exist, and adding support for new devices is straightforward. Sources for the SGI device translators supplied with the distributed system are maintained in the directory iraf$vms/gdev/sgidev. NOAO serves as a clearinghouse for new SGI plotter device interfaces; contact us if you do not find support for a local plotter device in the distributed system, and if you plan to implement a new device interface let us know so that we may help other sites with the same device.

The older NCAR kernel is used to generate NCAR metacode and can be interfaced to an NCAR metacode translator at the host system level to get plots on devices supported by host-level NCAR metacode translators. The host level NCAR metacode translators are not included in the standard IRAF distribution, but public domain versions of the NCAR implementation for VMS systems are widely available. A site which already has the NCAR software may wish to go this route, but the SGI interface will provide a more efficient and simpler solution in most cases.

The remaining possibility with the current system is the calcomp kernel. Many sites will have a Calcomp or Versaplot library (or Calcomp compatible library) already available locally.

To make use of such a library to get plotter output on any devices supported by the interface, one may copy the library to the hlib directory and relink the Calcomp graphics kernel.

A graphcap entry for each new device will also be required. Information on preparing graphcap entries for graphics devices is given in the GIO design document, and many actual working examples will be found in the graphcap file. The best approach is usually to copy one of these and modify it.

7.5.3. Image display devices

The majority of VMS/IRAF users will use a networked UNIX or VMS workstation running some version of the X window system (DECwindows/VMS or Motif/VMS in the case of VMS) for IRAF graphics and image display. X clients for graphics and image display are available for all IRAF platforms. See §6.1 of this manual for information about the *xterm* graphics terminal emulator and the *saoimage* image display server.

Those VMS/IRAF sites that have VAX/VMS workstations running the VWS display system can use the UISDISP.EXE display task in [IRAF.VMS.UIS] for image display. This is a standalone IMFORT program, i.e. it does not communicate with the tasks in the TV.DISPLAY package. See the file uisdisp.txt in the same directory for information on using the task.

Some interfaces for hardware image display devices are also available, although a general display interface is not yet included in the system. Only the IIS model 70 and 75 are currently supported by NOAO. Interfaces for other devices are possible using the current datastream interface, which is based on the IIS model 70 datastream protocol with extensions for passing the WCS, image cursor readback, etc. (see the ZFIOGD driver in iraf$vms/gdev). This is how all the current displays, e.g., imtool and saoimage, and the IIS devices, are interfaced, and there is no reason why other devices could not be interfaced to IRAF via the same interface. Eventually this prototype interface will be obsoleted and replaced by a more general interface.

If there is no IRAF interface for your device, the best approach at present is to use the IMFORT interface and whatever non-IRAF display software you currently have to construct a host level Fortran or C display program. The IMFORT library provides host system Fortran or C programs with access to IRAF images on disk. Documentation on the IMFORT interface is available in *A User's Guide to Fortran Programming in IRAF -- The IMFORT Interface*, Doug Tody, September 1986, a copy of which is included in the IRAF User Handbook, Volume 1A. If you do not have an existing image display program into which to insert IMFORT calls, it is not recommended that the TV.DISPLAY package code be used as a template. Rather, a standalone image display server should be constructed using the datastream protocol in the iraf$vms/gdev/zfiogd.x driver mentioned above (but that could be a very lengthy job; contact the Hotline).

8. Tuning Considerations

There are a number of things that can be done to tune VMS/IRAF for a particular host system:

- Install the executables to permit shared memory.
- Render the large runtime files contiguous to increase i/o efficiency.
- Precompile selected TERMCAP and GRAPHCAP entries.
- Strip the system to reduce disk consumption.

8.1. Stripping the system to reduce disk consumption

If the system is to be installed on multiple cpu's, or on a particularly small host like a MicroVAX, it may be necessary or desirable to strip the system of all non-runtime files to save disk space. This is done by deleting all the program sources and all the sources for the reference manuals and other printed documentation, but not the online manual pages. A special utility called *rmfiles* (in the SOFTOOLS package) is provided for this purpose. It is not however necessary to run rmfiles directly to strip the system. This may be done either from DCL or from within the CL.

```
$ set default [iraf]
$ mkpkg strip
$ set default [iraf.noao]
$ mkpkg strip
```

This will preserve all runtime files, permitting use of the standard runtime system as well as user software development. Stripping the system will *not*, however, permit relinking standard IRAF executables, as would be required for bug fixes or caching termcap and graphcap entries. The size of the system is reduced by about half. We do not recommend stripping the system libraries, as this would preclude all IRAF related software development or bug fixes, including user written Fortran programs (IMFORT), and we no longer provide a "*stripall*" option.

8.2. Installing executable images

The standard distribution of VMS/IRAF is configured to use a shared library for the bulk of the IRAF runtime system code. Use of this shared library considerably reduces the disk requirements of the system, while reducing runtime system memory usage and process pagein time, and speeding up process links during software development.

If the goal of minimizing physical memory utilization is to be achieved when running IRAF, it is essential that the IRAF shared library be "installed" in VMS system shared memory. Further memory savings through memory sharing can be achieved if some of the more commonly used IRAF executables are also installed. This becomes increasingly important as the number of IRAF users increases, but some memory savings may result even if there is only one user, since a single user may spawn batch jobs. Hence, the shared library s_iraf.exe should always be installed if IRAF is used at all, and cl.exe and x_system.exe are used by all IRAF jobs and should be installed if there is usually at least one IRAF user on the system. If there are usually several IRAF users on the system x_images.exe and x_plot.exe are probably worth installing as well.

Installation of task images in system shared memory is done with the VMS INSTALL utility, which requires CMKRNL privilege to execute (also discussed in §2.3.2). Installation of the IRAF shared library and other task images is most conveniently done by executing the INSTALL.COM command procedure in hlib. The list of files to be installed is in INSTALL.LST. This file should be reviewed and edited during system installation to select those images to be installed on the local host. Then INSTALL.COM may be directly executed to temporarily install the images, but to permanently install the images the system bootstrap procedure should be modified to execute the INSTALL.COM script (or explicitly install the images by name) whenever the system is booted. Note that the global pages used by an installed image are not actually freed until any and all processes using those global pages terminate.

The procedure for installing images will fail if there are insufficient global pages available in the system (for example, the number of global pages required to install the shared library in VMS/IRAF V2.10 is about 1267). If the system has insufficient free global pages to run the INSTALL.COM script one must either increase the number of system global pages (which requires rebooting the system), or one must edit the INSTALL.LST file to reduce the number of images to be installed.

8.3. Rendering large runtime files contiguous

The speed with which large disk files can be read into memory in VMS can degrade significantly if the disk is highly fragmented, which causes a large file to be physically stored in many small fragments on the disk. IRAF performance as well as VMS system throughput can therefore be improved by rendering frequently referenced large files contiguous. Examples of files which it might pay to render contiguous are the executables in `bin`, all of the runtime files in `dev`, and the large system libraries in `lib` (this last is only useful to speed software development, i.e., linking).

The easiest way to render a file contiguous in VMS is to use COPY/CONTIGUOUS to copy the file onto itself, and then purge the old version. Various nonstandard utility programs are available commercially which will perform this function automatically. The contents of an entire disk may be rendered contiguous or nearly contiguous by backing up the disk onto tape and then restoring it onto a clean disk.

8.4. Precompiling TERMCAP and GRAPHCAP entries

Precompilation of a termcap or graphcap entry is a technique used to speed runtime access of the entry for that device. If the entry is not precompiled the termcap or graphcap file must be physically opened and scanned at runtime to read the desired entry. This causes a slight delay when clearing the terminal screen or plotting a graph, hence it may be worthwhile to cache the entries for commonly used video and graphics terminals. We do not recommend that sites attempt to relink the IRAF system to cache new terminal device entries, but system managers may wish to be aware of this option.

The system comes with selected termcap and graphcap entries already precompiled. To see which devices are precompiled, page the cache data files, dev$cachet.dat (for termcap) and dev$cacheg.dat (for graphcap). To cache a different set of entries one must regenerate these files with the *mkttydata* task in the SOFTOOLS package, and then do a full sysgen relink with the *mkpkg* utility. Detailed instructions are given in the manual page for *mkttydata*.†

9. Software Management

9.1. Shared libraries

VMS/IRAF provides a shared library facility to reduce disk and memory requirements. What the shared library facility does is take most of the IRAF system software (currently the contents of the EX, SYS, VOPS, and OS libraries) and link it together into a special shareable image, the file `s_iraf.e` in the core BIN directory. This file is mapped into the virtual memory of each IRAF process at process startup time. Since the shared image is shared by all IRAF processes, each process uses less physical memory, and the process pagein time is reduced, speeding process execution. Likewise, since the subroutines forming the shared image are no longer linked into each individual process executable, substantial disk space is saved for the BIN directory. Link time is correspondingly reduced, speeding software development. For the shared library to be effective, it must be installed in system memory as described in §8.2.

9.2. Layered software support

An IRAF installation consists of the *core system* and any number of *external packages*, or "layered" software products. As the name suggests, layered software products are layered upon the core IRAF system. Layered software depends upon the core system for all of its functionality and is portable to any computer which already runs IRAF. Any number of layered products can be combined with the core system to produce the IRAF system used at a specific site. Due to disk space limitations it is likely that a given site will not wish to have all the available layered software products installed and on line at any one time.

†**Important Note**: If the system is configured to use the IRAF shared library (the default), you must update the system libraries (§9.7.2) and rebuild the shared library (§9.7.3) before relinking the system (§9.7.4), else the changes to the termcap and graphcap cache files will have no effect. The use of the shared library is peculiar to VMS/IRAF and is not discussed in the *mkttydata* documentation.

The support provided for layered software is essentially the same as that provided for maintaining the core system itself. Each "external package" (in most cases this actually refers to a tree of packages) is a system in itself, similar in structure to the core IRAF system. Hence, there is a LIB, one or more BINs, a HELP database, and all the sources and runtime files. A good example of an external package is the NOAO package. Except for the fact that NOAO happens to be located in the IRAF root directory, NOAO is equivalent to any other layered product, e.g., STSDAS, PROS, CTIOLOCAL, KPNOLOCAL, etc. Other layered products should be rooted somewhere outside the `[IRAF]` directory tree to simplify updates.

9.3. Software management tools

IRAF software management is performed with a standard set of tools, consisting of the tasks in the SOFTOOLS package, plus the host system editors and debuggers. Some of the most important and often used tools for IRAF software development and software maintenance are the following.

mkhelpdb	Updates the HELP database of the core IRAF system or an external package installed in hlib$extern.pkg. The core system, and each external package, has its own help database. The help database is the machine independent file helpdb.mip in the package library (LIB directory). The help database file is generated with *mkhelpdb* by compiling the root.hd file in the same directory.
mkpkg	The "make-package" utility. Used to make or update package trees. Will update the contents of the current directory tree. When run at the root iraf directory, updates the full IRAF system; when run at the root directory of an external package, updates the external package. Note that updating the core IRAF system does not update any external packages (including NOAO). When updating an external package, the package name must be specified, e.g., "mkpkg -p noao", and the command issued within the package directory, not from the IRAF root.
rmbin	Descends a directory tree or trees, finding and optionally listing or deleting all binary files therein. This is used, for example, to strip the binaries from a directory tree to leave only sources, to force *mkpkg* to do a full recompile of a package, or to locate all the binary files for some reason. IRAF has its own notion of what a binary file is. By default, files with the "known" IRAF virtual file extensions (.a, .o, .e, .x, .f, .h, etc.) are classified as binary or text (machine independent) files immediately, while a heuristic involving examination of the file data is used to classify other files. Alternatively, a list of file extensions to be searched for may optionally be given.
rtar,wtar	These are the portable IRAF tarfile writer (*wtar*) and reader (*rtar*) programs. These tasks produce archives compatible with the UNIX *tar* program. In addition they can move only the machine independent or source files (*wtar*, like *rmbin*, can discriminate between machine generated and machine independent files). A tarfile written on a VMS/IRAF system has the files blank padded, but *rtar* when used on a UNIX system will strip the trailing blanks by default.
xc	The X (SPP) compiler. This task can compile ".x" or SPP source files, knows how to link with the IRAF system libraries and the shared library, knows how to read the environment of external packages, and so on.

The SOFTOOLS package contains other tasks of interest, e.g., a program *mktags* for making a tags file for the *vi* editor, a help database examine tool, and other tasks. Further information on these tasks is available in the online help pages.

9.4. Modifying and updating a package

IRAF applications development (including revisions to existing programs) is most conveniently performed from within the IRAF environment, since testing must be done from within the environment. The usual development cycle is as follows. This takes place within the *package directory* containing all the sources and mkpkg-files for the package.

- Edit one or more source files.
- Use *mkpkg* to compile any modified files, or files which include a modified file, and relink the package executable.
- Test the new executable.

The *mkpkg* file for a package can be written to do anything, but by convention the following commands are usually provided.

mkpkg	Same as **mkpkg relink** below.
mkpkg libpkg.a	Updates the package library, compiling any files which have been modified or which reference include files which have been modified. Private package libraries are intentionally given the generic name libpkg.a to symbolize that they are private to the package.
mkpkg relink	Rebuilds the package executable, i.e., updates the package library and relinks the package executable. By convention, this is the file xx_foo.e in the package directory, where *foo* is the package name.
mkpkg install	Installs the package executable, i.e., renames the xx_foo.e file to x_foo.e in the global BIN directory for the system to which the package *foo* belongs.
mkpkg update	Does everything, i.e., a *relink* followed by an *install*.

If one wishes to test the new program before installing it one should do a *relink* (i.e., merely type "mkpkg" since that defaults to relink), then run the host system debugger on the resultant executable. The process is debugged standalone, running the task by typing its name into the standalone process interpreter. The CL task *dparam* is useful for dumping a task's parameters to a text file to avoid having to answer innumerable parameter queries during process execution. If the new program is to be tested under the CL before installation, a *task* statement can be interactively typed into the CL to cause the CL to run the "xx_" version of the package executable, rather than the old installed version.

When updating a package other than in the core system, the **-p** flag, or the equivalent PKGENV logical name, *must* be used to indicate the system or layered product being updated. For example, "mkpkg -p noao update" would be used to update one of the packages forming the NOAO system of packages.

The CL **process cache** can complicate debugging and testing if one forgets that it is there. Recall that when a task is run under the CL, the executing process remains idle in the CL process cache following task termination. If a new executable is installed while the old one is still in the process cache, the *flprcache* command must be entered to force the CL to run the new executable. If an executable is currently running, either in the process cache or because some other user is using the program, it may not be possible to set breakpoints under the debugger.

The **shared library** can also complicate debugging, although for most applications-level debugging the shared library is transparent. If it is necessary to debug IRAF system libraries, contact the IRAF Hotline for assistance.

A full description of these techniques is beyond the scope of this manual, but one need not be an expert at IRAF software development techniques to perform simple updates. Most simple revisions, e.g., bug fixes or updates, can be made by merely editing or replacing the affected files and typing

```
    cl> mkpkg update
```

plus maybe a *flpr* if the old executable is still sitting idle in the process cache.

9.5. Installing and maintaining layered software

The procedures for installing layered software products are similar to those used to install the core IRAF system, or update a package. Layered software may be distributed in source only form, or with binaries. The exact procedures to be followed to install a layered product will in general be product dependent, and should be documented in the installation guide for the product.

In brief, the procedure to be followed should resemble the following:

- Create the root directory for the new software, somewhere outside the IRAF directories.
- Restore the files to disk from a tape or network archive distribution file.
- Edit the core system file hlib$extern.pkg to "install" the new package in IRAF. Note that any '$' characters in a VMS pathname should be escaped with a backslash as in the examples. This file is the sole link between the IRAF core system and the external package.
- Configure the package BIN directory or directories, either by restoring the BIN to disk from an archive file, or by recompiling and relinking the package with *mkpkg.*

As always, there are some little things to watch out for. When using *mkpkg* on a layered package, you must give the name of the package being updated, plus the names of any other external packages used to build the target package, e.g.,

```
    cl> mkpkg -p foo update
```

where *foo* is the name of the package being update, e.g., "noao", "local", etc. The **-p** flag can be omitted by defining PKGENV as a VMS logical name.

Once installed and configured, a layered product may be deinstalled merely by archiving the package directory tree, deleting the files, and commenting out the affected lines in hlib$extern.pkg. With the BINs already configured reinstallation is a simple matter of restoring the files to disk and editing the extern.pkg file.

9.6. Configuring a custom LOCAL package

Anyone who uses IRAF enough will eventually want to add their own software to the system, by copying and modifying the distributed versions of programs, by obtaining and installing isolated programs written elsewhere, or by writing new programs of their own. A single user can do this by developing software for personal use, defining the necessary *task* statements, etc. to run the software in the personal login.cl file. To go one step further and install the new software in IRAF so that it can be used by everyone at a site, one must configure a **custom local package**.

The procedures for configuring and maintaining a custom LOCAL package are similar to those outlined in §9.5 for installing and maintaining layered software, since a custom LOCAL will in fact be a layered software product, possibly even something one might want to export to another site (although custom LOCALs often contain non-portable or site specific software).

To set up a custom LOCAL, start by making a local copy of the **template local** package that comes with the distributed system and editing hlib$extern.pkg to make the location of the new package known. If you make a source only *tar* or *backup* archive of iraf$local and install it as outlined in §9.4, you will have a custom LOCAL. The purpose of the template LOCAL is to provide the framework necessary for an external package; a couple of simple tasks are provided in the template LOCAL to serve as examples. Once you have configured a local copy of the template LOCAL and gotten it to compile and link, it should be a simple matter to add new tasks to the existing framework.

9.7. Updating the full IRAF system

9.7.1. The BOOTSTRAP

All current IRAF distributions come with the system already bootstrapped, i.e., the host system interface (HSI) comes with the HSI binary executables already built. A bootstrap should not be necessary unless one is doing something unusual, e.g., installing a bugfix or making some other revision to the HSI.

A bootstrap is like a full system sysgen, except that it is the host system interface (kernel and bootstrap utilities) which is compiled and linked, rather than IRAF itself. The system must be bootstrapped before a sysgen is possible, since the bootstrap utilities are required to do a sysgen. The two operations are distinct because only the bootstrap is machine dependent; everything else works the same on all IRAF systems.

The bootstrap utilities are required for system maintenance of the main system and hence must be linked first. However, unless a bug fix or other revision is made to one of the main system libraries (particularly LIBOS), there should be no reason for a user site ever to need to bootstrap the system. The bootstrap utilities are linked with the option /NOSYSSHARE, hence they should run on most versions of VMS.

The bootstrap utilities are written in C, and the DEC C compiler is required to compile or link these utilities. The C compiler is *not* required to run the prelinked binaries distributed with the system. To recompile and link the bootstrap utilities, i.e., to "bootstrap" IRAF, enter the commands shown below. Note that the HSI is always recompiled during a bootstrap; there is no provision for merely relinking the entire HSI (some individual programs can however be relinked manually by doing a "mkpkg update" in the program source directory).

```
$ set default [iraf.vms]
$ set verify
$ @rmbin.com
$ @mkpkg.com
```

The bootstrap should take 30 to 45 minutes on an unloaded 11/780. Once the bootstrap utilities have been relinked and installed in hlib, the main system may be relinked.

9.7.2. Updating the system libraries

Updating the system libraries, i.e., recompiling selected object modules and inserting them into the system object libraries, is necessary if any system source files are edited or otherwise modified.

The system libraries may be updated either from within the IRAF CL, or from DCL. For example, from within DCL:

```
$ set def [iraf]
$ mkpkg syslibs [mathlibs]
```

The brackets around mathlibs just mean "optional"; do not actually include the brackets if you need to rebuild the math libraries. This command will run the *mkpkg* utility, which will update each system library in turn, descending into the tree of source directories and checking the date of each object module therein against the dates of the source files and dependency files used to produce the object module, and recompiling any object modules which are out of date. In the worst case, e.g., if the system libraries have been deleted, the entire VOS (virtual operating system) will be recompiled. If it is known that no changes have been made to the math library source files, the optional mathlibs argument may be omitted to save some time.

9.7.3. Relinking the IRAF shared library

The IRAF shared library ([IRAF.BIN]S_IRAF.EXE) is constructed by linking the contents of the four main IRAF system libraries LIBEX, LIBSYS, LIBVOPS, and LIBOS. Hence, the system libraries must exist and be up to date before linking the shared library. The shared library must be installed in `bin` before it can be used to link any applications programs. At present, the shared library does not use transfer vectors, hence the full system must be relinked following any changes to the shared library. Note that since the shared library is normally installed in system memory, all installed IRAF images should be deinstalled and later reinstalled whenever the shared library is rebuilt. IRAF will not be runnable while this is taking place.

In the current release of IRAF the procedure for building the shared library has not yet been integrated into the automatic system generation procedure. Furthermore, utility tasks in the main IRAF system are currently used to build the shared library, so it is necessary to have at least part of the main system functioning before the shared library can be built. These problems (and the transfer vector problem) will be fixed in a future release of VMS/IRAF.

The shared library is rebuilt from within a running IRAF system that provides at a minimum the three executables CL.EXE, X_SYSTEM.EXE, and X_LISTS.EXE. If these executables do not yet exist or are not runnable, rebuild them as follows. Note the use of the "-z" link flag to link the new executables without the shared library.

```
$ set default [iraf.pkg.cl]
$ mkpkg update lflags=-z
$ set default [iraf.pkg.system]
$ mkpkg update lflags=-z
$ set default [iraf.pkg.lists]
$ mkpkg update lflags=-z
```

Now startup the CL and change to the directory hlib$share, which contains the code used to build the shared library.

```
$ set default [iraf.local]
$ cl
    (cl starts up...)
cl> cd hlib$share
```

The following commands will rebuild the shared library and install it in `bin`. This takes ten minutes or so.

```
cl> cl < makeshare.cl
    (takes a while)
cl> mkpkg install
cl> purge
cl> logout
```

The system libraries can be updated (§9.7.2) and the shared library rebuilt while IRAF is in use by normal users, however as soon as the command `mkpkg install` is executed (moving the new S_IRAF.EXE to bin) execution of normal user programs is at risk and one should kick all IRAF users off the machine and deinstall any installed IRAF images. The system may now be relinked and the newly linked shared library and images reinstalled, after which the system is ready for normal use again. Note that makeshare creates a load map s_iraf.map for s_iraf.e that should be deleted when no longer needed (it is quite large).

9.7.4. Relinking the main IRAF system

The following DCL commands may be entered to relink the main IRAF system, installing the newly linked executables in `bin`. Note that any IRAF executables that have been installed in system memory must be deinstalled and reinstalled (§8.2) following the relink.

```
$ set default [iraf]
$ mkpkg update
```

The command shown causes the full system to be relinked (or recompiled and relinked if any files have been modified) using the default compile and link switches defined in the global *mkpkg* include file hlib$mkpkg.inc. Any system wide changes in how the system is generated should be implemented by editing the contents of this file. For example, the default file supplied with the system includes the following line:

```
$set    LFLAGS  = ""            # default XC link flags
```

This switch defines the switches to be passed to the HSI bootstrap utility program *xc* to link the IRAF executables (no switches are given, hence the default link action will be taken). The switch "-z" may be included in the *xc* command line to link an executable directly against the system object libraries, rather than using the shared library. Hence, to relink IRAF without using the IRAF shared library (as is necessary for all but one system when there are multiple versions of IRAF simultaneously installed on the same machine) we would change the definition of LFLAGS as follows:

```
$set    LFLAGS  = "-z"          # default XC link flags
```

A full system relink takes approximately 30 minutes, or significantly less if the shared library is used. A full system compile and relink takes 9-24 hours depending upon the host system.

The above procedure only updates the core IRAF system. To update a layered product one must repeat the sysgen process for the layered system. For example, to update the NOAO package:

```
$ set default [iraf.noao]
$ mkpkg -p noao
```

Layered systems are completely independent of one another and hence must be updated separately.

10. Miscellaneous Notes

Magtape deallocation will not work properly in VMS/IRAF if the CL is run from a privileged account such as one that has BYPASS or SYSPRV privilege (a `cl> !dismount` may unmount the drive).

Under VMS 5, the AUTOGEN procedure includes feedback information. It is worth running the system, with IRAF users, for a couple of weeks and then re-running AUTOGEN (as detailed in the VMS System Management documentation) to adjust some of the system parameters (global pages, various memory structures) for the new work load.

"If all else fails", e.g., hung tape drives, etc., — our VMS system manager recommends kicking everyone off the system, perhaps even rebooting if it gets desperate, and then inspecting this Installation Guide again, before calling us. And please, if you need to call, try to have available the exact text of any error messages you may have encountered.

Appendix A. Private TMP and IMDIR Directories

If local system management policies require that private `tmp` and `imdir` directories be defined for each user, you can define these directories for each user as subdirectories of their SYS$LOGIN directory. One way to do this is define imdir to be the same as tmp, and modify the definition of IRAFTMP in the IRAFUSER.COM file as shown below.

```
$! ---------- add these lines to irafhlib:irafuser.com ----------
$! This block of DCL assigns the logical name IRAFTMP to a
$! subdirectory of the user’s VMS login directory.  It is necessary
$! to escape the "$" and "[" because they are IRAF filename
$! metacharacters.  This block of code should replace the current
$! definition in irafuser.com of IRAFTMP:
$!
$!   define/job  IRAFTMP  TEMPDISK:[IRAFTMP]        ! scratch files
$!
$! We assume only one occurrence of "$" or "[".
$!
$  tmpdir = f$logical ("SYS$LOGIN")
$  tmpdir = f$extract (0, f$locate("]",tmpdir), tmpdir) + ".IRAFTMP]"
$  define/job vmstmp ’tmpdir’
$  off = f$locate ("$", tmpdir)
$  tend = f$length (tmpdir)
$  if (off .ne. tend) then -
     tmpdir = f$extract (0, off, tmpdir) + -
       "\$" + f$extract (off+1, tend, tmpdir)
$  off = f$locate ("[", tmpdir)
$  if (off .ne. tend+1) then -
     tmpdir = f$extract (0, off, tmpdir) + -
       "\[" + f$extract (off+1, tend, tmpdir)
$  define/job iraftmp "’’tmpdir’"
```

In irafhlib:login.cl, replace U_IMDIR with "tmp$".

In irafhlib:mkiraf.com, replace the block of statements beginning "imdir_file :=" with the following:

```
$  if (f$search ("sys$login:iraftmp.dir") .eqs. "") then -
           create/directory vmstmp
```