VMS/IRAF Installation and Maintenance Guide

Doug Tody
Steve Rooke, Suzanne Jacoby, Nigel Sharp

National Optical Astronomy Observatories*
July 1987, Revised July 1989

ABSTRACT

Instructions are presented for installing, updating, and maintaining the VMS version of the portable IRAF system (Image Reduction and Analysis Facility). Step by step procedures are presented both for the initial IRAF installation and for updating an existing installation to the latest release of the system. Procedures for improving performance, minimizing disk and memory usage, and interfacing site dependent devices such as video or graphics terminals, plotters, image displays, and magnetic tape drives are discussed.

April 12, 1990

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

Historical Notes

The initial port of IRAF to VMS was carried out during 1984 and early 1985 at STScI (the Space Telescope Science Institute) by Jay Travisano and Fred Romelfanger, working under the supervision of Peter Shames, with some assistance early on from Jim Rose. The major system components implemented during this period were the SPP compiler (XC) and the OS interface routines for VMS, as specified in *A Reference Manual for the IRAF System Interface*, Doug Tody, May 1984.

Completion of VMS/IRAF took place at NOAO in the fall of 1985. This period saw the definition of a major new interface, the HSI (host system interface), and the implementation of the `mkpkg` system maintenance utility and many of the other HSI bootstrap utilities. This eliminated the final differences between the different versions of IRAF, allowing software to be routinely ported between UNIX/IRAF, VMS/IRAF, and so on, without any changes to the program sources or configuration control files, and allowing full automation of the system configuration control procedures.

The initial implementation of shared libraries for VMS/IRAF was carried out in the fall of 1986 by Dennis Crabtree of DAO (the Dominion Astrophysical Observatory, UBC, Canada), while on leave at STScI.

Contents

1. **Introduction** 1

2. **Initial System Installation** 2
   - 2.1. Create the IRAF Account 2
   - 2.2. Read the BACKUP Tape 3
   - 2.3. Edit the System Configuration Files 3
     - 2.3.1. [IRAF.VMS.HLIB]IRAFUSER.COM 4
     - 2.3.2. [IRAF.VMS.HLIB]INSTALL.COM 5
     - 2.3.3. [IRAF.VMS.HLIB.LIBC]IRAF.H 5
   - 2.4. Relink If Necessary 6
   - 2.5. Complete the Initial System Configuration 6
   - 2.6. Edit the System Dependent Device Files 7
     - 2.6.1. [IRAF.VMS.HLIB]ZZSETENV.DEF 7
     - 2.6.2. [IRAF.DEV]DEVICES 8
     - 2.6.3. [IRAF.DEV]TERMCAP 8
     - 2.6.4. [IRAF.DEV]GRAPHCAP 8
     - 2.6.5. IRAF Networking 9

3. **Updating an Existing IRAF Installation** 9
   - 3.1. Save Locally Modified Files 9
   - 3.2. Read the Distribution Tape or Tapes 10
   - 3.3. Merge Saved Files Back Into the New System 10
   - 3.4. Relink If Necessary 11
   - 3.5. Update VMS 11

4. **Testing a Newly Installed or Updated System** 12

5. **System Maintenance** 12
   - 5.1. Software Management 12
     - 5.1.1. Layered Software Support 12
     - 5.1.2. Software Management Tools 13
     - 5.1.3. Modifying and Updating a Package 14
     - 5.1.4. Installing and maintaining layered software 15
     - 5.1.5. Configuring a custom LOCAL package 16
     - 5.1.6. Bootstrapping the HSI 16
     - 5.1.7. Updating the System Libraries 17
     - 5.1.8. Relinking the IRAF Shared Library 17
     - 5.1.9. Relinking the Main IRAF System 18
   - 5.2. Tuning Considerations 19
     - 5.2.1. Installing Executable Images 19
     - 5.2.2. Rendering Large Runtime Files Contiguous 20
     - 5.2.3. Precompiling TERMCAP and GRAPHCAP Entries 20
     - 5.2.4. Stripping the System to Reduce Disk Consumption 20
   - 5.3. VMS Quotas and Privileges Required to Run IRAF 21
   - 5.4. Miscellaneous Notes 22

6. **Interfacing New Graphics Devices** 22
   - 6.1. Graphics Terminals 22
   - 6.2. Graphics Plotters 23

6.3. Image Display Devices 23

7. **The IRAF Directory Structure** 24

Appendix A. Private TMP and IMDIR Directories 26

Appendix B. Upgrading a Pre-IRAF 2.8 [iraf.local] Directory 27

Appendix C. Notes on Networking with DECNET in VMS/IRAF V2.8 31

VMS/IRAF Installation and Maintenance Guide

Doug Tody
Steve Rooke, Suzanne Jacoby, Nigel Sharp

National Optical Astronomy Observatories*
July 1987, Revised July 1989

1. Introduction

Installing VMS/IRAF is easy and fast, provided one takes the trouble to read and follow the instructions in this installation guide. Instructions are given both for the initial system installation (§2, immediately following) and for updating an existing installation (§3). Variations on the recommended system configuration are certainly possible and may be necessary at some sites due to restrictions or quotas, but not following the standard installation procedure or departing from the recommended system configuration may cause the system to function incorrectly, inefficiently, or not at all.

The distribution tape is a snapshot of our local (NOAO/Tucson) VMS/IRAF system. The device and system configuration tables come configured 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.

An installation typically takes about an hour, perhaps longer if relinking is necessary. Interfacing new graphics terminals, plotters, or image displays can take considerably longer, of course, and we will only touch upon such matters in the installation guide. 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 HOTLINE

telephone	`(602) 323-4160`	
internet	`iraf@noao.edu`	
span/hepnet	`noao::iraf`	(noao = 5355)
uucp	`{arizona,decvax,ncar}!noao!iraf` or	
uucp	`uunet!noao.edu!iraf`	
bitnet	`iraf@noao.edu` (through a gateway)	

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

2. Initial System Installation

It is only necessary to follow the full installation procedure if IRAF has not yet been installed on your system. If an old version of IRAF has already been installed, proceed to §3.

Briefly, the steps to be followed to install IRAF for the first time are the following. This summary is provided only to introduce the basic installation procedure: it is important to read the detailed installation instructions for additional details, to avoid certain errors or omissions which are otherwise highly likely (IRAF does not always follow VMS conventions).

- Login as SYSTEM and create an account for user IRAF, root directory `[IRAF]`, login directory `[IRAF.LOCAL]`, to be used by the IRAF system manager to maintain IRAF.
- Login as IRAF and run the `BACKUP` utility to restore the IRAF distribution tape or tapes to disk. Make sure the files belong to user IRAF.
- Edit the basic IRAF system configuration files `IRAFUSER.COM` and `IRAF.H` (in `hlib` and `hlib/libc`), e.g., to change the system dependent disk and batch queue names, and define the VMS pathnames to the major IRAF directories.
- Login again as SYSTEM and [1] copy the `IRAF.H` file edited above to the VMS system library directory `SYS$LIBRARY`, [2] add a global symbol `iraf` to the system-wide user login procedure (e.g. `SYLOGIN.COM`) which when executed by the user will execute the `IRAFUSER.COM` file to define the rest of the IRAF logicals, and [3] edit `INSTALL.LST` and execute the `INSTALL.COM` script (in `hlib`) to "install" in system shared memory at least the IRAF shared library, and probably some of the most frequently used executables as well.

Although the system installation is not yet complete, any user should now be able to run IRAF, i.e., run the `iraf` command procedure to define the IRAF logicals, run the `mkiraf` command procedure to initialize the IRAF environment, and type the command `cl` to start up the IRAF Command Language (CL). Following the basic installation procedure shown here, it will still be necessary to modify the IRAF device tables to tell the system about the local magtape devices, line printers, terminals, plotters, and so on. This should be done from within the running IRAF system to facilitate testing of the new device interfaces, and is an ongoing activity as new devices are interfaced to the system.

2.1. Create the IRAF Account

Create an account for user '`IRAF`'. This is recommended for the following reasons:

- All IRAF system management should be performed using the default protections etc. provided in the IRAF `login.com`.
- Multiple people may need to be IRAF system manager. Having a separate account avoids the need for one user to know another user's password. Even if there is only one site manager at your site, it may be necessary to give login information to the IRAF HOTLINE personnel to allow them to investigate a problem.
- Having IRAF owned by SYSMAN, SYSTEM or other management account is not a good solution as then anyone who needs to serve as IRAF site manager would require the manager's password, and furthermore, the protections etc. of a privileged account would cause problems for normal users.

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 (see §2.2 below). If necessary, the system can be stripped after installation to save space (§5.2.4), 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 login 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.
- 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 (§5.3).

2.2. Read the BACKUP 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. The tape contains approximately 8500 files in 300 directories, for a total of 48 Mb or 96000 512-byte blocks (including binaries). This disk size assumes a Disk Cluster Factor of 1; if your site has a larger DCF, IRAF will take more space (you can distinguish between allocated size and used size with `$dir/size=all`). The system as distributed uses a shared library to reduce disk and memory requirements. If IRAF is relinked and run without the shared library (e.g., if two or more versions of IRAF are installed on the same machine), the system will take up about 63 Mb or 125000 blocks.

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

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

2.3. 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.

---

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

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.

`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:[`*user*`]`. All potential IRAF users should have write permission and quota on this device. IRAF does not yet know about ACLs (access control lists).

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 yet understand 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 `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 and create a public directory `[IRAFTMP]` on this device. 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 "`tmp`" older than a day or so are garbage and should be deleted. It is best if "`tmp`" 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 `tmp` are rarely very large.

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

`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

---

†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.

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.

SYS$NODE

If you have DECNET installed, the system logical `SYS$NODE` will probably already be defined (`$ show logical sys$node`). If it is not defined, uncomment the line containing the sys$node definition, and set `NODE` to "::", as:

```
$  define/job SYS$NODE "::"
```

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 memory is necessary to permit memory sharing in VMS, and can greatly reduce physical memory usage and paging on a busy system. 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 §5.2.1 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 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 (i.e. `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††)*

---

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

††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.

These directory definitions are referenced only if logical name translation fails for some reason, 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. Relink If Necessary

If you received a full binary distribution of VMS/IRAF and you are not running an old version of VMS, you do not need to relink the system and may proceed to §2.5. A "you relink" distribution is shipped without binaries so of course requires linking. Linking may also be necessary when running a version of VMS older than that installed on our NOAO system when the executables in a binary release were linked, since the VMS operating system may refuse to run executables linked on a possibly incompatible "future" revision of VMS.

In any case, relinking the system is easy and fast. The basic procedure is outlined below; additional information is given in §5.1.9. The following commands should be executed while logged in as IRAF.

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

By default, the system is linked against the IRAF shared library `S_IRAF.EXE` which should already be present in the `[IRAF.BIN]` directory after restoring the distribution tape. This speeds up linking and considerably reduces the size of the resultant executables. It should not be necessary to relink the HSI executables (in `hlib`), as these are linked with `/NOSYSSHARE` on our system and are included in all VMS/IRAF distributions.

2.5. 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):†

```
$ 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.COM` file (or `SYSTARTUP_V5.COM`), 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.

This is separate from the user-level private `tmp` directory 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$COMMON:[SYSMGR]`. The `SYSTARTUP.COM` file (or `SYSTARTUP_V5.COM` in VMS V5.x), on the other hand, should go into `SYS$SYSROOT:[SYSMGR]` if the bootstrap procedure is different for each node.

```
$ @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 §5.2.1.

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.
```

At this point it should be possible for any user to run IRAF. First you can verify this by using the IRAF account itself; then individual users should set up their own IRAF startup directory by running `mkiraf`. The default `LOGIN.COM` in the IRAF login directory `[IRAF.LOCAL]` will automatically execute the `iraf` command to read the `IRAFUSER.COM` file and define the IRAF logicals. Type `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`). Typing `cl` next should cause the CL to run. If the CL runs successfully, the screen should be cleared (if the terminal 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.

2.6. Edit the System Dependent Device 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.

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

This file contains the name of the default editor, the default names of all the standard devices, and a number of other definitions which are not site dependent and which can therefore be ignored. To be accessible by the IRAF system, each local device named must also have an entry in the `TERMCAP` file (terminals and printers) or `GRAPHCAP` file (graphics terminals, image displays, and plotters) in `[IRAF.DEV]`. There must also be an "*editor*`.ED`" file in `[IRAF.DEV]` for the named editor; EDT, EMACS, and VI are currently supported (VI support is approximate). Edit the quoted value in the following entries to change the defaults. Sample values are shown.

```
set editor      = "vi"
set printer     = "imagen"
set terminal    = "vt640"
set stdgraph    = "vt640"
set stdimage    = "iism70l"
set stdplot     = "versatec"
```

For example, you may wish to change the default editor to "`edt`", the default printer to "`vmsprint`", the default image display to "`iism75`", and the default terminal to "`vt100`". Note that only a limited number of image displays is currently supported. Most video terminals, graphics terminals, and plotters are already supported by the current system. If you have

no device in a given class, you may wish to set the default to a nonexistent name, to generate a meaningful error message if someone should try to access it (e.g. `set stdimage = "nosuchdev"`).

2.6.2. [IRAF.DEV]DEVICES

This file contains a list of the allocatable devices (primarily tape drives) for the local system. It should be obvious how to change it by reading the comments in the file and studying the current values, which are for our system. By convention drives are assigned the logical names `mta`, `mtb`, and so on. The "`mt`" part is *required* for a magtape device. Note that in a VMS cluster where all nodes access the same version of IRAF, the tape drives for all systems must be defined in this one file, even though an individual drive may not be accessible on all nodes.

2.6.3. [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 it). A copy of the UNIX manual page documenting the termcap database is included with the installation kit in case you need to construct a new termcap entry for some device. Local additions should be placed at the top of the file to make it easier to merge the changes into a future IRAF release.

2.6.4. [IRAF.DEV]GRAPHCAP

There must be entries in this file for all graphics terminals, batch plotters, and image displays accessed by IRAF programs. Many graphics terminals are already supported; page the file to determine if entries are already available for the graphics terminals in use at your site, before attempting to write a new one. The IRAF file `sys$gio/doc/gio.hlp` contains a description of the graphcap database and instructions for adding an entry for a new device. A printed copy of this document is available upon request, however once IRAF is up you may find it easier to generate your own copy using `help` (the document is some 60 pages long), as follows:

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

The manual page for the `showcap` and `stty` tasks should also be printed since these utilities are useful for generating new graphcap entries. More focused documentation will be available eventually. Telephone consultation via the IRAF Hotline is available for those who need it. We ask that new graphcap entries be sent back to us so that we may include them in the master graphcap file for other sites to use.

---

†If the printer or plotter device is on a remote node which is not clustered with the local node but which is accessible via DECNET, 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. See Appendix C or contact us for assistance to help configure your system for IRAF networking.

2.6.5. IRAF Networking

The `dev` directory also contains the files (`HOSTS`, `HOSTLOGIN`, and `UHOSTS`), which are used by the IRAF networking software. Sites which have a local TCP/IP based network can enable IRAF networking by editing these files, substituting the hostnames and network addresses of the machines in the local network for the entries currently in the files, and by installing IRAF on the other nodes (or enough of IRAF to run the IRAF kernel server). It does not matter what native operating system runs on the remote nodes, so long as it runs IRAF as well.

We also provide limited support for DECNET based networking, but the current system as distributed must be relinked before DECNET networking can be used. Appendix C is included with notes on steps to be taken to implement DECNET networking, which should work in most cases. Be warned that only TCP/IP or DECNET style networking can be implemented, but not both. As this capability is still under development, please contact the Hotline for advice if you wish to use the IRAF DECNET interface.

3. Updating an Existing IRAF Installation

Skip to §4 if you have just completed the initial IRAF installation. This section is for sites upgrading to a new version of IRAF.

Updating is very similar to the initial installation (§2). The main difference is that material which has been added to the site dependent files since the initial installation should be saved and then merged back into the generic system from the distribution tape. The typical update process is summarized in the following steps and detailed further below:

- Save any site-specific files.
- Delete the old system.
- Restore the new system to disk from the distribution tape or network archive.
- Merge contents of saved site-specific files into new system.
- Relink the system if necessary.
- Update any "installed" images, the `<iraf.h>` file, etc.

Variations on this basic procedure are possible and may be preferred at some sites. For example, sites which make heavy use of IRAF and which have sufficient disk space may wish to install the new version of the system as a separate version on disk, e.g., calling the new version of the system the new ‘IRAF’ and keeping the old version around as ‘IRAFO’. At NOAO we maintain three separate versions of the system, IRAF (the current default user system), IRAFO (the old system, for backup), and IRAFX (the developmental system). Sites which wish to maintain multiple versions of the system should beware that only one version of the system can use the shared library, due to global section name conflicts. For efficiency reasons the default user version of the system ‘IRAF’ should use the shared library. The remaining versions of the system must be relinked (§5.1.9) to keep them from erroneously trying to use the shared library from a different release of IRAF.

3.1. Save Locally Modified Files †

---

†NOTE: Even if you had not modified the pre-2.8 `[iraf.local]` directory, you may want to save it. If you are installing IRAF 2.8 for the first time, with an earlier version of the system already present, it may be *very* important that you preserve the `[iraf.local]` directory, as a number of things have changed, and certain `local` tasks that were present in earlier releases are no longer supported in the standard system (e.g. `dsttoi, itodst, peritek, grinnell` tasks). This is not necessary if you do not plan to use these unsupported tasks from the pre-2.8 `local` directory.

Login as IRAF. 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 we delete the old system, below.

3.2. Read the Distribution Tape or Tapes

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 (e.g. if something were to have happened to the distribution tape en route). 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.

Now read the new distribution tape(s); it is important that this be done from the IRAF account, so the ownership and protections will be correct.

```
$ mount/foreign tape:
$ set default disk:[iraf]
$ backup/rew tape:iraf.bck/select=[iraf...] [...]
```

Instead of explicitly deleting the old system it is possible to read the distribution tape with `BACKUP/REPLACE`, but this really isn't any faster since a delete of each existing file is implied, and for those files or directories that have been renamed in the new release, junk files or directories will be left behind.

3.3. Merge Saved Files Back Into the New System

You can either merge the contents of locally modified files saved from the previous installation into their counterparts in the new system, or edit the new ones from scratch. Whichever is easier depends on how much editing you had to do in the first place, which probably depends upon the complexity of your local computer network and the number of peripheral devices. Any of the site configuration files might be modified in the distributed system from version to version of IRAF, so in general one cannot simply replace such a file with the one that was saved.

Merging means inspecting the two files (newly distributed vs. its saved counterpart) for differences, and deciding how to update the new file with local device names, directory pathnames, etc. The VMS `DIFFERENCES` utility may or may not be useful for this, depending on how much a file has changed, and how extensively it was modified locally. One thing that is a big help is to keep track of all local changes made to the standard system in a local "notes" file;

when the next system update occurs or when the installation is repeated on another cpu at the local site, one can then go down the list and make all the same changes to the newly installed system. By convention, that file would be called `notes.`*mysite*, in the `[iraf.local]` directory.

directory	*files*
`dev`	`devices, graphcap, termcap, devices.hlp` if present
`hlib`	`irafuser.com, install.lst`†, `zzsetenv.def, login.cl`
`hlib/libc`	`iraf.h` (→ `sys$library:iraf.h`
`local`	(see Appendix B)

The files which must be edited, via a diff/merge operation on the saved files or otherwise, are summarized in the table above. If you have added packages or tasks of your own to IRAF in versions prior to V2.8, you will probably no longer want to install them under the `[iraf]` directory tree, as facilities are now available for external layered packages; see §5.1.4. This may be deferred until the system is up and running if desired. The `[iraf.local]` directory is covered in Appendix B and in §5.1.5.

3.4. Relink If Necessary

If you received a fully binary distribution of VMS/IRAF and you are not running an old version of VMS, you do not need to relink and may proceed to the next section. See §5.1.9 for further information on relinking the system.

3.5. Update VMS

These next operations require system privileges, so login as SYSTEM. Perform the following series of operations:

- The `<iraf.h>` file (`[iraf.vms.hlib.libc]iraf.h`) may have changed in the new version of the system, so it should be recopied into the VMS system library `sys$library` after editing HOST, IRAF, and TMP for local pathnames as described in §2.3.3.
- The `install.com` procedure was changed at version 2.8 to allow for simple updating of the installed files. If you are already running IRAF 2.8 or later, and are now installing an even newer version, simply replace your old version of `install.lst` and re-run `install.com`. [Otherwise, if you have been running a version of IRAF prior to 2.8, then you should not replace your old version of `install.com`, but instead compare the set of executables you had installed in it to those in the new file `install.lst`. This latter file contains a list of all of the IRAF executables. Files which are *not* to be installed should be commented out, using a "!" at the beginning of the line. This list of installed files should agree with those from your old `install.com`.] Make sure that nobody on the system is running IRAF, and thereby locking the old IRAF executables, and then execute the new `INSTALL` procedure. This will clear away the old installed executables and install the new ones, paying as much attention as possible to clearing the global page structures.

In a VMS cluster, the shared images must be deinstalled and reinstalled on all nodes in the cluster which share the same version of IRAF. These steps are as discussed in §5.2.1; please refer to that section for additional information.

---

†See §5.2.1; if you are just now upgrading to IRAF 2.8, your old file will still be `install.com`, but you will want to edit only the new `install.lst`.

Lastly, logged 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.
```

4. Testing a Newly Installed or Updated System

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. Before starting the CL you should do what a regular user would do, i.e., run `mkiraf` to initialize the IRAF environment, and then edit the `login.cl` file created by `mkiraf` as desired. Any modifications you wish to preserve across another `mkiraf` should be placed into the 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 the template login.cl, but *must* have the statement `keep` as the last line.)

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

Once the IRAF environment is configured one need only enter the command '`cl`' to start up the CL. After a bit IRAF should print the message of the day and type out the root IRAF menu, then issue the "`cl>` " prompt indicating that it is ready for the first command. This assumes that the '`iraf`' command is executed in the `LOGIN.COM` file at login time; if this is not the case, the command '`iraf`' must first be entered from VMS to define `cl` and the other VMS logical names and symbols used by IRAF.

[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.]

```
$ cl
```

Once in the CL, you will probably have magtape and printer access, are likely to have graphics terminal access, and very possibly will not have either image display access or 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. You may also wish to run some of the benchmarks described in the benchmarks paper included with the installation materials, to make sure that your VMS system (user quotas and working set) is configured properly for efficient execution of IRAF.

5. System Maintenance

5.1. Software Management

5.1.1. 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.

The support provided for layered software is essentially the same as that provided for maintaining the core system itself. Each "external package" (usually this 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 rooted in the `iraf` 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.

5.1.2. 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 is analogous to the VMS `FORTRAN` except that it 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.

5.1.3. 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.

5.1.4. 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$extern.pkg` to "install" the new package in IRAF. Note that any `$` in a VMS pathname should be escaped with a backslash as in some of 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 product, you must give the name of the system being operated upon, e.g.,

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

where *foo* is the system or package name, e.g., "noao", "local", etc. The **-p** flag can be omitted by defining `PKGENV` as a VMS logical name, but this only works for updates to a single package.

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 of `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.

5.1.5. 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 §5.1.4 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$extern.pkg` to make the location of the new package known. If you make a source only `wtar` or `BACKUP` archive of `iraf$local` and install it as outlined in §5.1.3, 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.

5.1.6. Bootstrapping the HSI

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.

5.1.7. 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, as when using the `mkttydata` utility program to cache new termcap or graphcap device entries, or switching from TCP/IP to DECNET networking.

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.

5.1.8. 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 (V2.8) 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 login to the CL and proceed 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 (§5.1.7) 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.

5.1.9. 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 (§5.2.1) 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 `mkpkg.inc` in `hlib`. 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.

5.2. 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.

5.2.1. 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`. In IRAF 2.8 or later, 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 any currently installed images must be explicitly deinstalled before a new version can be installed (handled automatically in 2.8 by `install.com, install.lst`), and 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 V2.8 is about 1062). 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.COM` script to reduce the number of images to be installed.

5.2.2. 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.

5.2.3. 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 noticeable delay of as much as a second when clearing the terminal screen or plotting a graph, hence it is usually worthwhile to cache the entries for commonly used video and graphics terminals. It is generally not worthwhile for printers, plotters, and image displays.

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`.†

5.2.4. 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, caching termcap and graphcap entries, or switching to DECNET networking. The size of the system (the figures are for V2.8) is reduced

---

†**Important Note**: If the system is configured to use the IRAF shared library (the default), you must update the system libraries (§5.1.7) and rebuild the shared library (§5.1.8) before relinking the system §5.1.9), 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.

from about 48 Mb (megabytes) to around 25 Mb.† 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.

5.3. 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.

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 not use the 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

---

†Due to a bug in the stripping files in VMS/IRAF 2.8 not caught before the distribution tapes were cut, a `strip` will not delete quite as many files as it should (a few Mb). Contact the hotline for help in further stripping the system if this is necessary.

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 2.8, we have CL=324, S_IRAF=1062, X_SYSTEM=103, X_PLOT=118, and X_IMAGES=789. 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.

5.4. 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 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.

6. 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.

6.1. 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 §2.6.4). Assistance with interfacing new graphics terminals is available via the IRAF Hotline.

6.2. 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 included with the IRAF installation kit.

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.

6.3. Image display devices

The current IRAF system does not yet have a well defined and well isolated device independent image display interface. Work on such an interface is currently underway; contact the IRAF group at NOAO for further information on the status of the new display interfaces. Further information on the image display interfaces currently available may be found in the *IRAF Newsletter*.

Those VMS/IRAF sites that have VAX/VMS workstations running the VWS display system (e.g. VAXstation) may run the `NEWUISDISP.EXE` display task directly in `[IRAF.VMS.UIS]`. This is a standalone IMFORT program, i.e. it does not communicate with the tasks in the `tv$display` package. See the file `newuisdisp.txt` in the same directory for help.

Sun workstations running `IMTOOL` and `GTERM` may be used as front ends to VAXes that use the default TCP/IP networking. See the `imtool(1)` manual pages on the Sun.

Likewise, workstations running the MIT X window system may also be used as front ends, using the *ximage* display server †. This was developed for IRAF by CfA (SIAO), but is distributed by the IRAF project; a distribution kit is available upon request. We are looking forward to the release of X11/NeWS by Sun sometime during 1989, and will provide X11 based versions of gterm and imtool for Sun/IRAF once a vendor supported version of X11 is available for the Sun (the X11 versions of imtool and gterm should run on any manufacturer's X11 based

†At press time for IRAF 2.8, `ximage` was only available for X10, with an X11 version under development.

workstation). Contact the hotline for advice.

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 `vms/gdev`). This is how all the current displays, e.g., imtool and ximage, 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 `vms/gdev/zfiogd.x` driver mentioned above (but that could be a very lengthy job; contact the hotline).

A few sites may have the Gould IP8400 or IP8500 displays; code for the Gould is not automatically supplied with VMS/IRAF, and must be requested separately. The interface is not very satisfactory, but can be used in a pinch. (If you are installing IRAF 2.8, the Gould interface has not changed since version 2.5, so if you have an old version, you will not need to request a new one.)

7. The IRAF Directory Structure

A graph of the current full VMS/IRAF directory structure is given at the end of this document. The main branches of the tree are summarized below. Beneath the directories shown are some 300 subdirectories, the largest directory trees being `sys`, `pkg`, and `noao`. The entire contents of all directories other than `vms`, `local`, and `dev` are fully portable, and are identical in all installations of IRAF sharing the same version number.

`bin`	- the IRAF BIN directories
`dev`	- device tables (`termcap`, `graphcap`, etc.)
`doc`	- assorted IRAF manuals
`lib`	- the system library; global files
`local`	- iraf login directory; locally added software
`math`	- sources for the mathematical libraries
`noao`	- packages for NOAO data reduction
`pkg`	- the IRAF applications packages
`sys`	- the virtual operating system (VOS)
`vms`	- the VMS host system interface (HSI = kernel + bootstrap utilities)

The contents of the `vms` directory (host system interface) are as follows:

`as`	- assembler sources for VMS/IRAF
`boot`	- bootstrap utilities (mkpkg, rtar, wtar, etc.)
`gdev`	- graphics device interfaces (SGI device translators)
`hlib`	- host dependent runtime files
`mkpkg.com`	- executed to bootstrap the VMS/IRAF HSI
`os`	- OS interface routines (VMS/IRAF kernel)
`rmbin.com`	- executed to delete binary files in subdirectories
`uis`	- UIS display program for VAX workstations

If you will be working with the system much at the system level, it will be well worthwhile to spend some time exploring these directories and gaining familiarity with the system.

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
```

Appendix B. Upgrading a Pre-IRAF 2.8 [iraf.local] Directory

If you are installing IRAF 2.8 for the first time on a system which had an earlier version of IRAF, *and* you either had your own tasks in [iraf.local] or you want to use the unsupported tasks formerly distributed in local (dsttoi, itodst, peritek, grinnell), you will need to change a few things. If you did not make local additions or use those unsupported tasks, or if you are installing a later version of IRAF than 2.8, you may skip the rest of this section.

Starting at IRAF 2.8, there is support for external layered packages, which will largely free you from having to merge local additions into future updates. The new local directory is set up as a *template* external package, and it is best that it remain that way. True local additions (plus the former, unsupported tasks mentioned above) are best placed in a directory of your choice outside the IRAF tree. The external layered package mechanism is discussed in §5.1.4 and 5.1.5, to which you should refer for background information now, prior to completing this section. The instructions below are intended as a guide and may not be enough for all sites, depending on the complexity of the local additions. The [iraf.noao] directory may be used as an additional guide. This all sounds a bit complex, but it will be worth it because it only need be done once, and all you will need to do in future IRAF releases is edit hlib$extern.pkg to hook up your local package.

Briefly, you will need to create your own local directory, preferably outside the IRAF tree, e.g. [kpnolocal], (*yourlocal* used for explanatory purposes below). Copy the *new template* [iraf.local] directory into it:

```
$ set default [yourlocal]
$ backup [iraf.local...] [...]
$ delete login.*;*, bugs.*;*, notes.*;*, irafks.*;*
$ delete $forward.;*, *readme.;*, telinit.;*, [.uparm]*.*;*
$ set protection=(owner:wd) uparm.dir
$ delete uparm.dir;*
```

This is a summary of the steps needed to upgrade a pre-2.8 local directory to use the new layered package mechanism; an example follows:

- In what follows it is assumed you want your new package to be known only as local; the pathname for local defined in [iraf.vms.hlib]extern.pkg will take care of its location.
- Copy the source, parameter, and object files and libraries from your saved copy of the old [iraf.local] directory tree (as saved in §3.1) into [*yourlocal*.src] or subdirectories, and any executables into [*yourlocal*.bin]. Help files (.hlp) should go into [*yourlocal*.src.doc].
- In [*yourlocal*.src] edit x_local.x to add your tasks, and edit the mkpkg file along the lines of the templates.
- Edit the package local.cl, .hd, and .men files in [*yourlocal*.src].
- Edit [iraf.vms.hlib]extern.pkg to replace the iraf$local/ pathname with your VMS specific one as in the commented-out example for kpnolocal, escaping any $ in VMS pathnames with a backslash.
- Execute MKPKG from [*yourlocal*] to build the package ($ mkpkg -p local update).
- After logging into the CL, execute MKHELPDB to install your package and task help pages into IRAF from the package directory, giving lib/root.hd and lib/helpdb.mip as the task parameters.

When finished, the next time a user logs into the CL, the local package will appear

listed in the main package menu, help will be available, and the tasks will be installed.

Example: Add a brand new task to the new `local` package.

Let's add a simple "hello world" task to a new copy of the template local package in the directory `[ourlocal]` on disk `scr$2`. Log into the IRAF account.

```
$ create/directory scr$2:[ourlocal]
$ set default scr$2:[ourlocal]
$ backup irafdisk:[iraf.local...] [...]
$ delete login.*;*, bugs.*;*, notes.*;*, irafks.*;*
$ delete $forward.;*, *readme.;*, telinit.;*, [.uparm]*.*;*
$ set protection=(owner:wd) uparm.dir
$ delete uparm.dir;*
$ edit irafhlib:extern.pkg
```

Change the pathname of the local directory:

```
      reset local = scr\$2:[ourlocal]
```

```
$ set default sys$login
$ cl
cl> cd local$src
cl> ed x_local.x
```

```
      task      bswap               = t_bswap,
                pavg                = t_pavg,
                hello               = t_hello


cl> ed hello.x
      procedure t_hello

      bool      verbose
      bool      clgetb()

      begin
                verbose = clgetb ("verbose")

                if (verbose)
                    call printf ("Hello world!  (the long version)\n")
                else
                    call printf ("Hello\n")
      end
```

```
cl> ed hello.par
      verbose,b,h,no,,,"use the long form of hello?"
cl> ed local.cl
```

```
      #{ Package script task for the LOCAL package.  Add task declarations here
      # for any locally added tasks or subpackages.

      cl < "local$lib/zzsetenv.def"
      package local, bin=localbin$

      task      bswap,
                pavg,
                hello = "local$src/x_local.e"

      clbye()
cl> ed local.hd
      # Help directory for the LOCAL package.

      $defdir           = "local$src/"
      $doc              = "local$src/doc/"

      bswap             hlp=doc$bswap.hlp,src=bswap.x
      pavg              hlp=doc$pavg.hlp,src=pavg.x
      hello             hlp=doc$hello.hlp,src=hello.x
cl> ed local.men


      bswap - Swap the bytes in a file
       pavg - Plot the average of the lines of an image or image section
      hello - Hello world demo task
cl> cd doc
cl> ed hello.hlp
```

```
        .help hello Jul89 local
        .ih
        NAME
        hello - demo task
        .ih
        USAGE
        hello verbose
        .ih
        PARAMETERS
        .ls verbose
        Use verbose=yes to test the parameter mechanism.
        .le
        .ih
        DESCRIPTION
        This is a demo task to assist in adding local additions to the template
        local package.
        .ih
        EXAMPLE
        1. Use the long version

                 cl> hello verbose+
        .ih
        BUGS

        .ih
        SEE ALSO
        .endhelp
cl> cd local
([ourlocal])

cl> softools
so> mkpkg -p local update
so> mkhelpdb lib/root.hd lib/helpdb.mip
so> bye
cl> local
     bswap hello pavg

lo> hello

Hello

lo> hello verbose+

Hello world!  (the long version)

lo> help hello
(etc.)
```

Appendix C. Notes on Networking with DECNET in VMS/IRAF V2.8

IRAF networking is the preferred way to gain access to peripherals on a remote computer. These peripherals might include printers, plotters, disks, image display devices and possibly magnetic tape drives. IRAF networking can be used to access peripherals on nodes in a local area network, where the networking protocol in use is internally supported by IRAF. We presently distribute IRAF configured for TCP/IP networking. A DECNET networking interface is also available for VMS nodes and is the focus of this document. The IRAF/DECNET network interface does not, at present, permit access to tape drives on a remote machine [contact the hotline for advice; there may be a workaround for this before long]. If tape drive access is needed, it will be necessary to perform a normal IRAF installation on the remote node and use the tape reading facilities on that machine directly (once installed, the system may be stripped considerably).

If your system uses only DECNET, it may or may not be necessary to use IRAF networking, depending upon what peripherals it is desired to access on the remote node. Installing IRAF networking with a minimal 'kernel server' implementation is easy to set up. An alternative to this approach may be to supply DCL command procedures IRAF can use to send printer and plotter output to the remote node without using IRAF networking. Contact us to find out if this will work on your system.

The default network protocol on the distributed systems is TCP/IP, because that is what we use in the local NOAO network. In its present form, the choice of network protocol is a compile-time option. In IRAF V2.8 the appropriate object file is supplied automatically (unless you have a source-only distribution or have stripped all object files from the system), but you still need to rebuild the shared library and perform a relink sysgen.

There are two ways to use internal IRAF networking: between two fully installed IRAF systems, and between one fully-installed system and a 'kernel server' system. The steps below under 'Configuring IRAF networking for DECNET' apply to a fully-installed system, and would have to be taken for at least one node in an IRAF network. See the section below under "Server Nodes" if you are using the remote purely for access to peripheral devices, but not for running IRAF. In node name references in these notes we will use the convention HOMENODE = fully-installed system, and SERVERNODE = the server-only node.

VMS/IRAF networking using DECNET continues to be supported in a limited way in IRAF V2.8. These notes should be sufficient for you to successfully install IRAF/DECNET networking, but don't hesitate to call the HOTLINE if questions arise. We have not tested these procedures in all possible environments, so they may be inaccurate or incomplete for some configurations. There are at least two special situations to be aware of:

[1] If you use "rooted logicals" for the IRAF root directory on any of your nodes, you will need to get some modified networking code from us.

[2] If your system management policies prohibit establishing a logical link connection with a remote task addressed as object type 0, you will not be able to use IRAF/DECNET networking without a possible workaround from the hotline. The task specification string used in the networking code is of the form:

```
NODE"USERNAME PASSWORD"::"TASK=IRAFKS"
```

[3] An additional note for users rebuilding the VMS/IRAF shared library is that when you remake the shared library (`makeshare.cl`) on a MicroVAX, warning messages may appear from the linker. These are harmless, and are due to the fact that on a MicroVAX (as opposed to other VAXes) the system services show up in a load map as virtual addresses (`7FFF****`). You can modify `hlib$share/makeshare.cl` so it filters these addresses out of the load map generated when rebuilding the shared library. The

warnings are harmless, but can be avoided by editing `makeshare.cl` so the section of the script that filters the addresses using the IRAF match task reads as follows:

```
# Now do the UNIVERSAL symbols
# Select out those psects with attributes like Fortran Common blocks

match (pattern = "-R", files = "common.map") | words |
        match (pattern = "^00", stop=yes) |
        match (pattern = "^7FF", stop=yes) |
        match (pattern = "^CC_", stop=yes) |
        sort > symbol.tab
```

I. Configuring IRAF networking for DECNET on a fully-installed system

This is a summary of the steps you will take:

- build a decnet version of `zfioks.obj` and load it into a library
- rebuild the shared library
- relink all the IRAF executables
- relink irafks.e without the shared library
- edit `dev$hosts`
- create `.irafhosts` files in IRAF home directories on home node
- create `irafks.com` files in VMS login directories on server node

[1] Get into the directory `os$net` (`[iraf.vms.os.net]`). Check to see if there is a file `zfioks_obj.dna`. If there is not, go to step [2]. If there is, copy it:

```
$ copy zfioks_obj.dna zfioks.obj
```

[2] (There was no `zfioks_obj.dna` file in `[iraf.vms.os.net]`, so we will attempt to create one.)

If you do not have a DEC C compiler, contact us to see about getting a copy of `zfioks.obj` built for DECNET; you will not be able to proceed further until you do.

You do have a DEC C compiler, so edit `zfioks.c` and compile it:

```
$ edit zfioks.c
```

change from:

```
#define DECNET  0
#define TCP     1
```

to:

```
#define DECNET  1
#define TCP     0
```

```
$ cc/nolist zfioks.c
```

[3] Load the zfioks object file into the IRAF kernel library:

```
$ lib/replace [iraf.vms.hlib]libos.olb zfioks.obj
$ delete zfioks.obj;*
```

[4] Make sure the system is configured for IRAF networking. Examine the file `[iraf.vms.hlib]mkpkg.inc`, and make sure `USE_KNET = yes`:

```
$set USE_KNET = yes          # use the KI (network interface)
```

[5] Assuming you are using the shared library, the default, you now need to regenerate that library. If for some reason you are not using the shared library, you may proceed to step [6]. To rebuild the shared library, carry out the steps in section 5.1.8 then proceed to step [6].

[6] Relink the main IRAF system. To do this, carry out the instructions in section 5.1.9 of the Installation Guide. Make sure you have deinstalled all IRAF executables before relinking. A common mistake is to have the old shared library installed while trying to relink with the new version. The iraf kernel server executable (`bin$irafks.e`) should NOT be linked against the shared library. So, after you have relinked the IRAF executables (including `irafks.e`) with the new shared library, go back to the `iraf$sys/ki` directory and relink the kernel server without the shared library:

```
$ xc -z irafks.o
```

Replace the `irafks.e` currently in `iraf$bin` with this version.

[7] Edit two files in `dev$`:

`hosts`	set up node names for your site
`hostlogin`	(optional) set up default public logins if any

A common error is to edit the `dev$hosts` file incorrectly. Make sure you add an entry for each node, including the local node, to this file.

[8] Create `.irafhosts` files in the IRAF login directories of users who plan to use networking on HOMENODE. The file format is a series of lines of the form:

```
alias1 alias2 ... aliasN : loginname password
```

If the same login name and password are used on several nodes, an "*" may be used as the alias. The file should be read protected; also, a "?" may be used in place of the password (you will be prompted). For example:

```
# .irafhosts file
node1 node3 : mylogin mypassword
* : otherlogin ?
```

[9] Place a DCL command file called "`irafks.com`" in the VMS login directories of all IRAF network users on SERVERNODE (or on both nodes for two-way network traffic); note the editing required after `"$! If this is a standalone..."`, if the target node is a fully configured IRAF system — the default assumes the target is a kernel server only system:

```
$! IRAFKS.COM -- DCL command file to run the IRAF/DECNET kernel server.
$! Note:  the path to the server directory MUST be fully specified
$! below (i.e. substitute your actual disk name for "DISK").
$!
$! set verify!*debug*
$! netserver$verify = 1!*debug*
$!
$! If this is a standalone irafks server,...
$  set default DISK:[irafserve]       ! comment out if full IRAF
$  @irafuser.com     ! comment out if full IRAF
$  run irafks.exe     ! comment out if full IRAF
$! ..., else if this is a fully-configured IRAF system,...
$! iraf     ! <- uncomment if full IRAF
$! run irafbin:irafks.exe     ! <- uncomment if full IRAF
$  logout
```

where `DISK` is of course the IRAF disk; the decnet server has no access to globally defined IRAF symbols or logicals like `IRAFDISK` until it can execute `irafuser.com`.

This completes the basic installation; the steps are the same for both nodes if two-way IRAF network traffic is desired.

II. Server Nodes

If a remote node is to be used purely for access to peripherals (you don't plan actually to run anything in IRAF on the remote node directly), you need only install a small subset of the IRAF system on it. This is a summary of the steps you will take:

- build a version of `irafks.exe` without the shared library
- install and edit the following files in `[irafserve]` on the server node:

```
irafks.exe
irafuser.com
irafks.com
login.com
zzsetenv.def
```

[1] On the existing fully-installed IRAF system, HOMENODE, verify that the kernel server executable in `irafbin` is indeed the version you just created, linked without the shared library. You will be copying it to the SERVERNODE in a following step.

[2] On the remote, or 'kernel server' system, SERVERNODE, create an IRAF account as was done for the the main IRAF installation, as in section 2.1 of the Installation Guide. Although it is not strictly necessary to have an actual account for IRAF on this machine, it would be helpful should it ever be necessary for us to debug anything on it. Have its default directory set to `[IRAFSERVE]` rather than `[iraf.local]`, and create the directory `[irafserve]` (this could be put elsewhere; you would simply change all references to `[irafserve]` in these notes).

Also have the system manager insert the following global symbol definition into the system-wide login file (e.g. `sys$manager:sylogin.com`):

```
$  irafserve :== @DISK:[irafserve]irafuser.com
```

If this is not done, then each user must explicitly execute that '@' command in their own `login.com` files.

[3] On the remote, or 'kernel server' system, SERVERNODE, log in as IRAF. You should be in the `[irafserve]` directory. Copy the newly-created kernel server executable and some other files from the fully-installed system onto the kernel server system. The file `irafks.com` was discussed in step [9] above; if you do not have one on HOMENODE, just create it here.

```
$ set def [irafserve]
$ copy HOMENODE"account password"::DISK:[iraf.bin]irafks.exe []
$ copy HOMENODE"account password"::DISK:[iraf.vms.hlib]irafuser.com []
$ copy HOMENODE"account password"::DISK:[iraf.vms.hlib]zzsetenv.def []
$ copy HOMENODE"account password"::DISK:[iraf.local]login.com []
$ copy HOMENODE"account password"::DISK:[iraf.local]irafks.com []
```

with the appropriate substitutions of course.

Edit `irafuser.com` to set the logical name definitions `IRAFDISK, IMDIRDISK`, and `TEMPDISK` for the server node.

```
$ edit irafuser.com          (etc.)
```

Make sure the first executable lines in `login.com` are as below:

```
$ edit login.com
```

```
$  set prot=(s:rwd,o:rwd,g:rwd,w:r)/default
$  if (f$mode() .eqs. "NETWORK") then exit
```

Change the DISK specification for the server system in `irafks.com` to be the actual disk (or a system-wide logical name for it) -- see step [9] above.

```
$ edit irafks.com(etc.)
```

[4] Insert `irafks.com` files into the VMS login directories on the server node of any IRAF users wanting access to this node. If plotter access is desired, it is essential that the users' `login.com` files explicitly execute the `[irafserve]irafuser.com` file, and desirable that they exit if mode = `NETWORK` (to avoid lots of error messages in `NETSERVER.LOG` files). For example (in user's `login.com`):

```
$  set prot=(s:rwd,o:rwd,g:rwd,w:r)/default
$  if (f$mode() .eqs. "NETWORK") then exit
$  irafserve      ! define IRAF logicals for plotting
```

(The global symbol "`irafserve`" was defined in the system-wide login file in step [2] above.)

This completes the basic network configuration. Although setting up plotting devices in a network environment can be complicated, here are some guidelines for installing SGI translators on a kernel server node.

III. Guidelines for installing SGI translators on kernel server node

This is a summary of the steps needed:

- on home node, edit `dev$graphcap` (and `dev$hosts` if necessary)
- copy `hlib$sgiqueue.com` to server and edit it
- copy `hlib$sgi2XXXX.exe` to server (one for each XXXX SGI translator)

[1] On the home node, edit the graphcap file (`dev$graphcap`) to insert the server node prefix into the device name part of the `DD` string. For example, if device `vver` lives on SERVERNODE, and the `DD` string for that device begins with:

```
:DD=vver,tmp$sgk,...
```

change it to:

```
:DD=SERVERNODE!vver,tmp$sgk,...
```

with appropriate substitution for SERVERNODE. If all the plotters to which you want access are on the same node, you can use the `plnode` alias; just make sure `plnode` is an alias for SERVERNODE in `dev$hosts` (see 'Configuring IRAF networking for DECNET' above). Also make sure the printer queue names in the `DD` string are appropriate for SERVERNODE or "`none`" if the devices are not spooled. E.g.:

```
:DD=plot!vver,tmp$sgk,submit/que=fast/noprint/nolog \
/para=\050"vver","$F","$(PX) $(PY) VERSAQUEUE"\051 \
irafhlib\072sgiqueue.com:tc=sgi_versatec:
```

[2] Copy the home node's `hlib$sgiqueue.com` file to the `[irafserve]` directory on SERVERNODE and edit it for the local printer queue names, etc. Also replace the "`irafhlib`" in the VMS task definitions, as the translators on SERVERNODE reside in the same directory as `sgiqueue.com`, i.e. `[irafserve]`. E.g.:

```
$  sgi2vver := $DISK:[irafserve]sgi2vver  !  versatec interface
```

[3] Copy the home node's SGI translators to the `[irafserve]` directory on SERVERNODE.

There are many different ways of interfacing plotters in VMS environments, so these notes will not work in all cases. Call the IRAF Hotline if you have any trouble at all [(602) 323-4160].

IV. Using the network

Each user desiring IRAF network access should have the following files, as described above:

on HOMENODE: `.irafhosts` in IRAF home directory
on SERVERNODE: `irafks.com` in VMS login directory
on SERVERNODE: "`irafserve`" command in VMS `login.com` file

In some cases, for example where the local node is a MicroVAX with comparatively slow disk drives and the remote is a big VAX with very fast disk drives, an image may actually be accessed faster on the remote over the IRAF network than it can be locally.

When working on images across the network, it is best to store the pixel files in the same directory as the image headers, or in a subdirectory. This is done by "`cl> set imdir = HDR$`". Or, to cut the size of the directory in half by placing the pixel files in a subdirectory called "`pixels`", "`set imdir = HDR$pixels/`". To cut down on typing, you might also define a logical directory for your data area on the remote, "`dd`" for example. Thus you could place the following lines in your `login.cl` file:

```
set imdir = HDR$pixels/
set dd    = bigvax!mydisk:\[mydir.iraf.data]
```

Suppose you are now in the CL on the HOMENODE, and you want to run `IMPLOT` on an image called `test001` on the remote node BIGVAX, in directory `[mydir.iraf.data]`, and then copy it to your current directory on node A:

```
pl> implot dd$test001
pl> imcopy dd$test001 test001
```

The logical directory `dd$` may be used in most of the normal fashions (you won't be able to "`chdir`" to it). Both IRAF virtual file names may be used (including all the usual system logical directories) and host file names. As usual in VMS/IRAF, any "$" and "[" in host-system pathnames must be escaped with "\", and lowercase filenames should only be used in IRAF virtual filenames. Various file access examples follow:

Use of environment variable (see above) to cut down on typing:

```
cl> set dd = bigvax!mydisk\$1:\[mydir.iraf.data]
cl> dir dd$
cl> page dd$README
im> imhead dd$*.imh
```

Use of IRAF virtual filenames explicitly: (this will only work on systems where bigvax is a fully-installed system; the IRAF virtual directories may not be present on a server-only system)

```
cl> count bigvax!local$notes
cl> tail bigvax!local$notes nl=100 | page
cl> dir bigvax!hlib$*.com long+
cl> dir bigvax!sys$gio/R* long+
cl> page bigvax!sys$gio/README
cl> page bigvax!home$login.cl
cl> dir bigvax!tmp$sgk* long+
```

Use of host-system pathnames:

```
cl> dir bigvax!usr\$0:\[mydir.iraf]*.imh
cl> copy bigvax!usr\$0:\[mydir.doc]letter.txt newletter.txt
cl> history 100 > bigvax!usr1:\[mydir]test.his
```

Please remember that this is a developing interface, and that it has not yet been tested as extensively as we would like. We would appreciate being contacted about any problems you uncover.