IRAF Version 2.10 Revisions Summary

IRAF Group
National Optical Astronomy Observatories†
July 1992

ABSTRACT

A summary of the revisions made to the IRAF for the version 2.10 release are presented. V2.10 is a major IRAF release, meaning that there were significant additions to both the system and applications software, and the release will eventually be made available on all supported systems. This document deals with only the changes to the IRAF core system and NOAO packages. The many layered packages available for IRAF follow independent release schedules and are documented separately. This revisions summary is divided into three main sections: system revisions, IRAF and NOAO applications revisions, and revisions to the IRAF programming environment.

July 4, 1992

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

Contents

IRAF Version 2.10 Revisions Summary

IRAF Group
National Optical Astronomy Observatories†
July 1992

1. Introduction

This document summarizes the changes made to IRAF in version 2.10. V2.10 is a major release of IRAF, meaning that there were significant changes to both the system and applications software, and the release will eventually be made available on all supported platforms.

By IRAF version 2.10 we refer only to the core IRAF system and NOAO packages. Numerous external or "layered" packages are also available for IRAF, for example the NSO package (solar astronomy), the ICE package (data acquisition), the STSDAS package from STScI (HST data reduction and analysis), the TABLES package from STScI (tabular data), the XRAY package from SAO (X-ray data analysis), and so on. These packages are layered upon the IRAF core system, and once installed, are indistinguishable from the rest of IRAF. Layered packages are developed and maintained separately from the core IRAF release and follow independent release schedules, hence we will not discuss them further here. Contact the IRAF project or any of the outside groups supporting IRAF layered packages for additional information on what is available.

This document is intended only as an overview of what is new in version 2.10 IRAF. More detailed documentation will be found in the systems and applications notes files (files `sysnotes.v210.Z` and `pkgnotes.v210.Z` in the network archive), in the online help pages, and in any reference papers or user's manuals distributed with the software. The IRAF Newsletter is a good source of information for new IRAF software.

The reader is assumed to already be familiar with the basic concepts and operation of the IRAF system. In particular, familiarity with V2.9 IRAF is assumed.

2. IRAF System Revisions

2.1. Starting up V2.10 IRAF

Before attempting to start V2.10 IRAF each user should run the *mkiraf* command in their IRAF login directory. This will create a new `login.cl` file and `uparm` (user parameter) directory. *mkiraf* should be allowed to delete any existing parameter files, as there have been many changes to task parameter sets in the new version of IRAF.

2.1.1. LOGIN.CL changes

The `login.cl` file is read by the CL during startup to perform some runtime initialization and to customize IRAF for each user. A standard `login.cl` file is created and initialized by the *mkiraf* command when the user's IRAF login directory is configured. For V2.10 IRAF the `login.cl` file has undergone the following changes.

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

- The IRAF version number is now checked automatically whenever you login, and a warning message will be printed if your `login.cl` file is out of date. If you see this message it means that important changes have been made to the `login.cl` file and you need to rerun *mkiraf* to update this file.
- Most of core IRAF system packages are now loaded automatically at login time by the `login.cl` file. If you use a personal `loginuser.cl` file and you previously loaded any core system packages in this file, you should edit the file and remove those entries.
- A "quiet" login option is now provided. If a file named `.hushiraf` exists in the login directory when you start up the CL, printing of the usual login messages will be disabled (the only output seen will be the "cl>" prompt).
- On UNIX/IRAF systems the login script now looks at the host system environment variable `TERM` and checks for the common terminal types "sun" and "xterm", configuring the IRAF *stty* accordingly if either terminal type is seen (note that the default number of lines set for an xterm terminal window may need to be modified). The logic used to do this is not foolproof however, and is provided only as an example illustrating how to make use of the host system terminal definitions. You may need to customize this portion of the script, or override it in your `loginuser.cl` file.
- The CL hidden parameter `showtype` is now set to "yes" by default. This will cause a character to be printed after the name of each package or named pset in CL package menus to allow these objects to be easily distinguished from ordinary tasks. Packages are marked with a trailing period (".") and psets with an ampersand ("@"). This feature can be disabled with a "showtype=no" statement.
- The V2.10 login script contains a call to a new internal (non-user) IRAF task *mtclean*. Be sure to leave this alone, it is required for the correct operation of the new magtape i/o system.

The USER package defined in the template `login.cl` has been extensively revised, adding many new tasks. These are mainly used to make common UNIX commands available from within the IRAF environment. Type "?user" in the CL to see what is available, e.g.:

```
cl> ?user
     adb     cp      fc      lpq     mv      rlogin  spell   top
     bc      csh     find    ls      nbugs   rsh     sps     touch
     buglog  date    finger  mail    nm      rtar    strings vi
     cal     df      ftp     make    od      ruptime su      w
     cat     diff    generic man     ps      rusers  sync    wc
     cls     du      grep    mkpkg   pwd     rwho    telnet  wtar
     comm    emacs   less    mon     rcp     sh      tip     xc
```

Note that since the USER package is defined in the user's login file it is easily customized to add new tasks. Refer to the existing package for examples illustrating how to do this.

2.1.2. Compatibility Issues

Version 2.10 IRAF requires the new `login.cl` file; if the CL does not start up correctly, it may be because the user has not done a *mkiraf*, or because they have some construct in their `loginuser.cl` file which is incompatible with V2.10 IRAF. The V2.10 login file is usable with V2.9 IRAF, however this is not recommended.

There have been many task **parameter changes** between V2.9 and V2.10. If "parameter not found" messages are seen, most likely the user has an old `uparm` directory, or has been switching back and forth between IRAF versions. An *unlearn* or *mkiraf* should fix the problem.

The V2.10 IRAF **networking system** is not fully compatible with earlier versions of IRAF. This can cause problems when, e.g., a newly installed V2.10 system is used to communicate with an older version of IRAF on another system. The best solution is to update to V2.10 on all systems, but if this is not convenient it is possible to configure the networking system to avoid the problems. See the discussion of the new networking system given below.

Accessing a **remote magtape device** via IRAF networking will not work between V2.10 and older versions of IRAF (the remote procedure calls have changed). To remotely access magtape devices you will need to install V2.10 IRAF on both the client and server nodes.

In most respects installing V2.10 IRAF will be very similar to installing earlier versions of IRAF. The main difference is the **tapecap file** required to use the new magtape system. The old `dev$devices` file is no longer used. See the discussion of the new magtape system given below for more details.

Due to name changes in certain low level system routines (made to avoid name clashes when linking with host level libraries) the V2.10 libraries are incompatible with older versions of IRAF. Any IRAF programs or external packages **relinked** under V2.10 will have to be fully recompiled or the linker will complain about unresolved externals. Note that so long as the old program is not relinked there should be no problem, even if the program uses the IRAF shared image, since the V2.9 shared image is included in V2.10 (this applies to Sun/IRAF systems only).

Starting with V2.10, many IRAF applications now fully support generalized **world coordinates** (WCS). While in principle this should not pose any compatibility problems, the image headers do contain more information in V2.10 than previously, and there can be problems if, for example, an input image contains an illegal WCS. Previous versions of IRAF would ignore this but in V2.10 such an image could result in an error or warning message. If WCS related problems are encountered it is probably best to contact the IRAF group for help.

2.2. CL Enhancements

2.2.1. Formatted scans and prints, scan from a pipe

New in the V2.10 CL (command language) are formatted scan and print routines, and the ability to scan from a pipe or other form of redirected input. These new facilities will prove most useful in CL scripts.

The old unformatted scan and print routines are the following. These are still available and are the simplest routines to use where they are adequate.

```
  scan (arglist)          # scan standard input
 fscan (list, arglist)    # scan a list
 print (expr, exprlist)   # print to standard output
fprint (param, exprlist)  # print to a string buffer
```

For example,

```
list = "filename"
while (fscan (list, x, y) != EOF)
    print ("x=", x, "y=", y)
```

In the above, *arglist* is a comma delimited list of output arguments (parameter or parameter field names) and *exprlist* is a comma delimited list of expressions to be printed. *list* is the name of a list-structured parameter to be scanned, and *param* is the name of a parameter, the value field of which is to receive the output string. The unformatted scan routines will automatically convert output data values to match the types of the output arguments.

The new formatted routines are as follows. These take an extra *format* argument which tells how to parse the input string in the case of the *scanf* routines, or how to format the output in the case of the *printf* routines.

```
 scanf (format, arglist)        # formatted scan from stdin
fscanf (list, format, arglist)  # formatted scan from a list
printf (format, exprlist)       # formatted print to standard output
```

Currently there is no *fprintf* routine. For the *printf* routine the *format* argument is similar to that for the SPP/VOS *printf* (which is similar to the C *printf*). The *format* argument for the *scanf* routines is the same as the VOS LIBC *scanf*, which is patterned after the C *scanf* (in fact the UNIX manual page for *scanf* can be used as a guide to the CL *scanf* with only minor deviations).

The following examples illustrate the new routines.

```
cl> printf ("%d foo %7.3f\n", 5, 12.123) | scanf ("%d foo %g", i, x)
cl> printf ("new values are i=%d, x=%g\n", i, x)
new values are i=5, x=12.123
```

or,

```
while (fscanf (list, " %*d size=%d name=%s", i, s1) != EOF)
    printf ("size=%05o, name='%s'\n", i, s1)
```

Note in the first example the use of *scanf* to scan from a pipe. There are actually two different versions of *scan* and *scanf* in V2.10 IRAF, an intrinsic function version and a procedure version. When called as an intrinsic function, a *scan* routine returns as its function value the number of operands successfully scanned, or EOF. When called as a procedure, the function value of a *scan* routine is discarded.

Here is another example illustrating scan from a pipe, in this case using an unformatted scan since the *hselect* output is in a simple tabular format (*hselect* prints selected fields of the image header).

```
cl> hselect dev$pix naxis,naxis1,naxis2 yes | scan (i, j, k)
cl> printf ("naxis=%d, axlen=[%d,%d]\n", i, j, k)
naxis=2, axlen=[512,512]
```

When using the formatted scan routines, care must be taken to ensure that the data types implied by the *format* argument match the actual data types of the output parameters. The *scanf* routines are implemented using an internal call to the C (LIBC) *scanf*, with the output parameter value fields referenced directly via a pointer. If the data type is incorrect the output value may be meaningless.

2.2.2. Unlearning package parameters

The *unlearn* task now works for package parameters as well as task parameters. In a command such as "unlearn pkgname" the package parameters for the named package will be unlearned, as well as the parameters for all the tasks in the package. This works whether or not the package is loaded.

2.2.3. Loading packages at login time

A bug has been fixed which affected packages with package parameters loaded at login time. It is now permissible to load any package at login time regardless of whether it has package parameters (V2.9 users will recognize this bug as one which prevented loading CCDRED in the login script).

2.2.4. Environment variables

The environment variables `imtype`, `cmbuflen`, and `min_lenuserarea` are now defined at login time. Previously, explicit values for these variables were not defined, and the system would use the builtin internal defaults. Explicit definitions were added so that the user can query the current value, e.g.

```
cl> show cmbuflen
128000
```

A *show* or *set* with no arguments will print the full environment list. New values for these and other common environment variables may be set in the user `login.cl` file.

2.3. System Management Related Changes

2.3.1. Install script

The UNIX/IRAF install script underwent minor changes to make it more robust. Problems are still possible if the IRAF root pathname is set to different values in the various system dependent files modified by the script. The system as shipped from NOAO has the same initial root pathname set in all such files, but problems can occur if the files are manually edited during or after installation. To avoid problems always use the install script to make system changes such as installing at a different root directory.

2.3.2. Caching of termcap entries

User caching of termcap or graphcap entries with the old `mkttydata` task is no longer recommended. The most common entries (e.g. sun, xterm, vt100) are already cached. Modern workstations are so fast that there is no longer much point in caching termcap entries; it is sufficient to merely place local additions near the top of the file. Most programs that repeatedly access the terminal cache the entries internally during execution. Custom caching of termcap or graphcap device entries requires that the system be relinked, and the risk inherent in relinking the system (hence giving up the prebuilt, pretested binaries) is not worth the small performance gain achieved.

2.3.3. Sorting of UNIX directories

The UNIX-based versions of IRAF now sort UNIX directories whenever a directory is accessed to expand a file or image template. This will fix the problem sometimes seen in earlier versions of IRAF, in which an image template could appear to be expanded in a seemingly random fashion.

2.3.4. UMASK support

The UNIX-based versions of IRAF now support the host level *umask* file creation mask correctly. If files or directories created by V2.10 IRAF do not have the desired permissions, it is because you do not have umask set correctly at the UNIX level (most people set umask to 022).

2.4. Networking Enhancements

2.4.1. New networking driver

The UNIX/IRAF networking driver has been completely rewritten for version 2.10 IRAF, with the goals of eliminating redundant password prompts, improving efficiency, and enhancing system security. For the most part the changes will be transparent to the user. Once the IRAF system manager has configured the `dev$hosts` file for the local site the networking system should function automatically; in the default configuration a password prompt should be seen only when connecting to a server for which *rhosts* ("trusted" hosts) permission is not granted.

The following information is provided mainly for IRAF system managers. In normal use the user does not need to understand how the networking system functions.

2.4.1.1. How it works

The IRAF networking system is an RPC (remote procedure call) mechanism for the IRAF kernel; all kernel procedures may execute either locally or remotely, and the client and server nodes do not even need to run the same operating system. IRAF applications may be distributed, and may access resources which reside anywhere on the network. IRAF networking is layered upon standard low level networking protocols such as TCP/IP and DECNET.

The IRAF networking system defines one or more *connection protocols* which are used by a client to connect to the IRAF kernel server on a remote machine. The old networking driver supported only one connection protocol, the *rexec* protocol, which requires a login name and password. The new driver adds support for an *rsh* based protocol. This is the default connection protocol for V2.10 IRAF; automatic fallback to the rexec protocol is provided in the event that the rsh connect fails. The rsh connection protocol bootstraps off the suid-root *rsh* command found in most BSD derived UNIX systems (most System V systems provide the equivalent command *remsh*).

The connection protocol is used to start the *in.irafksd* IRAF networking daemon on the remote server node. This daemon executes with the same uid and permissions as the account which initiated the connection, and there is one such daemon per user per server node. Once the daemon has been started via the rsh or rexec connection protocol, new client connections are made very quickly, by merely forking the daemon to create the IRAF kernel server process, and setting up a direct socket connection between the IRAF client process and the server. The daemon process runs indefinitely, shutting down if idle for longer than a certain interval (the current default is one hour). When connecting to the daemon a client must supply an authentication key to gain access to the daemon. If authentication fails the daemon shuts down and it is necessary to reestablish the connection.

2.4.1.2. The .irafhosts file

The new networking driver retains the old *irafhosts* file, used to store information telling how to connect to various IRAF hosts (the irafhosts file is the file `.irafhosts` in the user's login directory). The networking system will automatically create this file for the user if the file is not found; if an old-style file is found, it will be edited by the system to make it compatible with the new networking system. While it is no longer necessary for the irafhosts file to contain password information to avoid password prompts, the file is used to store the user authentication key, hence the file should be read protected. The networking system will automatically read protect the file if it is not already protected.

To avoid authentication failures when clients on different nodes attempt to connect to the same server, the same authentication code should be used on all server nodes. Unfortunately there is no way that the networking system can do this automatically (without going to some much more complicated authentication scheme, such as a key server), so users who make heavy use of the networking system should install a copy of their irafhosts file in their login directory on all server nodes. If this is not done the networking system will still work, but will be less efficient than it could be, when simultaneously accessing the same server from IRAF sessions running on multiple client nodes.

The truly paranoid may not be happy with even the unique user authentication code used in the current networking system. If this is the case the *port* parameter (see below) may be set to zero to force rsh to be used for every connection (in effect the in.irafksd daemon has to be restarted for every connection). This imposes an overhead of as much as several seconds on every server connect. Alternatively, `KSAUTH` can be defined in the user environment at login time, setting the value string to some random integer value selected at login time. If defined in the user environment, `KSAUTH` will override the value of *auth* given in the irafhosts file. This approach would at least allow efficient connects for a single login process tree.

The irafhosts file consists of two sections. The first section defines several networking parameters: `port`, `auth`, `hiport`, and `timeout`. The second section is a list of server nodes, with login and password information describing how to connect to each node.

```
port = default
auth = 1234567890
hiport = default
timeout = default

ursa    : <user> ?
*       : <user> <user>
```

The example above illustrates a typical irafhosts file. Typically a unique authentication code is allocated automatically by the system when the file is first created, and the other parameters retain their default values as shown (i.e., the string "default"). In the example the host list consists of an entry for the node "ursa", and an entry for everything else. The format of a host entry is "*host-name : login-name password*". If *login-name* is the reserved string "<user>" the user name on the client node is used for login authentication on the remote node. Setting the password to "<user>" as well causes the rsh connect protocol to be used; anything else causes the rexec protocol to be used. If the rexec protocol is used the password field may be set to the actual password or to the string "?", in which case the system will prompt for the password whenever a connection attempt is made. The "*" entry should always be the last entry in the list, since it matches all nodes. The default host list contains only the "*" entry.

Additional information on the irafhosts file is provided in the comments in the file `dev$irafhosts`, and in the system notes file.

2.4.1.3. Compatibility

By default the new networking system will try to use the rsh protocol to connect to the server node. If the server is running an older version of IRAF the connection attempt will hang and eventually time out. If this occurs the networking system will fall back on the rexec protocol and issue a password prompt, but by then the user will probably have interrupted the connect. The best way to avoid this problem is to update the server node to V2.10, but if this is not possible, an explicit entry for the node can be made in the irafhosts file, setting the password field so that the rexec protocol will be used.

An older, e.g. V2.9 client connecting to a V2.10 server should not be a problem. In this case the V2.10 server will see an attempt to connect with the rexec protocol, which should be processed as in the past.

Aside from the problem of making a connection the pre-V2.10 and V2.10 networking systems are compatible, *except* for the magtape i/o calls. Since the magtape driver calling sequences were changed in V2.10, remote magtape access between V2.10 and older systems is not supported.

2.4.2. The hosts file

The file `dev$hosts` is used to interface new host machines to the IRAF networking system. This file defines, for each host, the primary host name, any aliases, and the command to be executed remotely to start up the IRAF kernel server on a remote node.

The format and role of the hosts file is unchanged in V2.10. Two changes were made which affect the use of this file.

- A user can now have a private copy of the hosts file. To enable this feature, the variable `irafhnt` (IRAF host name table) should be defined in the user's IRAF or host level environment, with the string value giving the file pathname of the user's private host name table file.

- The maximum number of entries in the host name table has been increased from 64 to 128. In the current implementation these entries are statically buffered, so it is necessary to keep the maximum number of entries to a reasonable value.

The hosts file must be configured to enable IRAF networking. IRAF networking is disabled if there is no entry for the local host in this file. The *netstatus* command may be used to examine the state of the host name table after it has been loaded by the networking system.

There is another file `dev$uhosts` which often confuses people. This file is not used by UNIX/IRAF and can be ignored; it is there for VMS/IRAF and other IRAF implementations which do not provide the equivalent of the UNIX system routine *gethostbyname*. On host machines which implement name server facilities IRAF will use the name server, allowing any machine on the internet to be accessed via IRAF networking, so long as there is an entry in the system wide or user IRAF hosts file.

2.5. Magtape System Enhancements

The magtape subsystem underwent a major revision in V2.10. The VOS level MTIO interface was extensively revised, and the host-level IRAF magtape driver ZFIOMT is completely new. A new device configuration facility called *tapecap* was introduced. Together with the new "programmable" magtape driver, this makes it possible to interface almost any removable media mass storage device to the magtape subsystem. The DATAIO applications, in particular the FITS programs, underwent minor, but important revisions.

A full specification of the new magtape subsystem, particularly the tapecap facility, is beyond the scope of this document. Our intention here is merely to introduce the new facilities. A reference paper is planned, time permitting, which will fully document the new magtape system and tapecap.

2.5.1. Basic usage

In most respects basic usage of the magtape system is unchanged from previous releases. Many new capabilities have been added, but these are upwards compatible with earlier releases.

2.5.1.1. Logical device names

Magtape devices are still referred to in IRAF applications much as they have been in the past. Whether or not the logical device names are the same before and after the V2.10 upgrade depends upon how the IRAF system manager configures the tapecap file. The new magtape system is much more flexible than previously regarding device names, but to avoid user confusion it is recommended that the old names be supported as aliases regardless of whatever the full device name may be.

As in earlier versions of IRAF, a simple magtape reference might be

```
cl> mtexamine mta
```

where "mta" is the device name. To examine only file 3 on the tape one might enter

```
cl> mtex mta[3]
```

If the device is on the remote node "ursa" the command would be

```
cl> mtex ursa!mta[3]
```

If the device is a 9 track tape supporting multiple densities we might specify the density explicitly, e.g.

```
cl> mtex mta1600[3]
```

Device names can be more complex. For example,

```
        cl> mtex mtwd
```

might refer to a WangDAT drive, and

```
        cl> mtex mtwdc
```

to the same drive, but with data compression enabled.

Devices can have multiple names, possibly with slightly different behavior or characteristics. Device names such as "mta" are usually only host specific aliases for the lower level, host independent device names. The names "mta" and "mtb" might be aliases for the actual device names "mthp0" and "mtxt1". This can be useful in networked systems where IRAF and the tapecap file reside on a server node, but the user is running IRAF on their workstation with a local tape drive attached. In this case there may be no entry for the local drive in the installed tapecap file, but the user can still access the local drive using the lower level, host independent device entry (it is also possible to have a private tapecap file as we shall see later).

To see what logical devices are known to the magtape system you can enter the following command (the task *gdevices* was intended for graphics devices but can be pressed into service to list a tapecap file as well). Just because a device is listed does not mean that the physical device actually exists on a particular host system.

```
        cl> gdev devices='^mt' graphcap=dev$tapecap
```

In IRAF V2.10 it is possible to include tapecap parameters in the device specification to do clever things, as in the following example.

```
        cl> mtex "mta[:so=lepus:se:nb]"
```

This is discussed further below in the section describing the tapecap facility.

2.5.1.2. Device allocation

Device allocation operates much the same in V2.10 as in earlier versions of IRAF. The main change is that it is no longer necessary to allocate a device in order to be able to access it. It is strongly recommended however that you always allocate a device before accessing it in IRAF. If the device is not allocated anyone can access the drive, e.g., changing the tape position, and this can cause data to be lost or improperly read back. The only reason to access the drive without allocating it is if there is some problem with device allocation on your system.

A device is allocated with the *allocate* command, e.g.

```
        cl> alloc mta
```

A device is deallocated with *deallocate*. If the tape has already been unmounted use the *rewind=no* option to avoid accessing the drive. By default the tape will be rewound when it is deallocated. Deallocating and reallocating a drive initializes the magtape system, i.e., all cached knowledge of the status of the drive is discarded.

2.5.1.3. Device status

The device status can be examined at any time that the magtape device is idle (not being actively accessed by an IRAF task) using the *devstatus* task.

```
        cl> devs mtc
        # Magtape unit mtc status Thu 12:54:02 04-Jun-92 user v14
        file = 4
        record = 1
        nfiles = 0
        tapeused = 405
        pflags = 0
```

Of particular interest are the *tapeused* and *nfiles* fields. *nfiles* refers to the total number of files on the tape (if a file is appended to the tape it will be file *nfiles*+1). If *nfiles* is given as zero that means that the magtape system does not yet know how many files are on the tape (it has not seen the end of tape).

tapeused reports the amount of tape used in units of kilobytes. This is intended to refer to the amount of tape used up to and including the end of tape (EOT). However, the magtape system only knows about data that it has seen, i.e. physically read or written, so whether or not *tapeused* is accurate depends upon how you have accessed the tape.

For example, if you started out with a fresh tape and appended a number of files the number should be accurate. If you just completed a full scan of the tape with *mtexamine* the number should be accurate, since all the data was read. If you just put on a new tape and did a scan of the FITS headers with "rfits make-", the number may or may not be accurate, depending upon the device and how file skipping forward was done. In most cases only the header area of each file will have been read and *tapeused* will not reflect the full contents of the tape. If however, "rfits make-" is done immediately after writing to a new tape, the number will be accurate, since all the data was written before the tape was rescanned to print the FITS headers.

Be advised that by default an explicit *rewind* will clear the *nfiles* and *tapeused* fields, since by default *rewind* initializes the magtape system. See the discussion of *rewind* below.

In summary *tapeused* can be useful for monitoring how much space is left on a tape, but you have to know what you are doing. When writing to a new tape the number will be accurate so long as you avoid doing an explicit *rewind*. A simple procedure which will always work when mounting a non-empty tape and appending files to it, is to do an *mtexamine* of the tape and then append the new files. This can be time consuming for large tapes but does provide a safe and device-independent method for getting the maximum amount of data on a tape.

2.5.1.4. File positioning

File positioning when accessing magtape files in IRAF is straightforward in the sense that you can simply specify the file number to read an existing file, or "append" (as in wfits new-) to append a file to an existing tape. Most tasks accept range lists to access existing files, e.g.

```
cl> mtexamine mta file="3,5,9-11"
```

It is even possible to position a tape to a specific record within a file. Opening a tape with file zero (as in "mta[0]") disables file positioning, allowing the tape to be positioned with host level utilities to workaround media problems.

There can be problems with this simple scheme however, with modern tape devices such as DAT and Exabyte which have capacities in the gigabyte range and which may be used to store thousands of files. It is beyond the scope of a revisions summary to go into this in detail here, but a simple example will help illustrate the problem.

Assume we have a tape mounted on device "mtwd" containing 900 files. We want to append image "pix" as a FITS file. The obvious thing to do is enter the following command.

```
cl> wfits pix mtwd new-
```

This will certainly work. The only problem is that if the tape is freshly mounted the magtape system will not know how many files there are on the tape, and it will have to skip forward one file at a time to count the files and determine where EOT is. In the worst case of a tape containing several thousand files this could literally take hours.

If we know apriori the number of files on the tape, e.g., 900 in our example, the following command would be much faster (something like 10-40 seconds on most DAT drives, assuming a decent host level driver).

```
cl> wfits pix mtwd[901]
```

Of course, if there were actually 930 files on the tape, the last 30 files would be overwritten.

There is a useful trick which works in some cases if we don't care exactly how many files are on the tape (whether this works depends upon the specific device and the host driver).

Assume that we know there are several hundred files on the tape, but less than 1000. We enter the following command to append a file to the tape.

```
        cl> wfits pix mtwd[1000]
```

If this works, after the operation the magtape system will think there are 1000 files on the tape. A subsequent "wfits new-" would be very fast regardless of the tape position, since so long as the magtape system knows where the end of tape is relative to the current position, it can get there very fast.

If the trick doesn't work for your particular device or driver you will probably get a positioning error when attempting to position to a nonexistent file beyond the EOT.

More automated techniques for rapid positioning with very high capacity tapes are still a matter for study. One promising technique would be to formalize the above approach by supporting EOT-relative positioning. A tape catalog based approach is also possible. The best approach may simply be to not write so many small files on large tapes, by grouping images or other data system files into a single large tape file (as with UNIX *tar*). None of these approaches are implemented in V2.10.

2.5.1.5. Rewind

In IRAF a magtape device is rewound with the *rewind* command, as in the following example.

```
        cl> rewind mta
```

By default this will not only rewind the tape but also initialize the magtape system, in the sense that all cached information regarding the named device is erased (e.g., *nfiles* and *tapeused* in the cached device status). This is necessary when changing tapes without deallocating the drive; always do an explicit rewind (or deallocate) of the device before removing a tape or mounting a new one. Having *rewind* initialize things is also useful if error recovery should fail following an interrupt or program abort.

In some cases it may be useful to be able to do a rewind without erasing the cached device status. This is done by specifying the *init*- option on the command line.

2.5.2. New magtape driver

The IRAF magtape driver is new for V2.10. The chief feature of the new driver is that it is programmable, via the tapecap device entry, making it possible to interface new devices or host drivers without having to make any binary code changes to IRAF. All one has to do is add a device entry in the tapecap file.

2.5.2.1. Software structure

The IRAF magtape system has enough layers that it may be confusing exactly what the magtape driver is and what role it plays. A brief review of the software structure may help clarify things.

Starting at the top we have applications, such as the DATAIO and MTLOCAL tasks, which can access magtape files. These use the IRAF/VOS interfaces FIO (file i/o) and MTIO (magtape i/o) to do i/o to tape devices. For the most part i/o is done with FIO and is device independent, but a call to the magtape system is required to open a magtape device. The tapecap file is read by the GTY interface, which is called by MTIO. MTIO passes the tapecap device entry as a string to ZFIOMT, the IRAF magtape device driver, when a tape file is opened. All magtape positioning and i/o is done by ZFIOMT driver under the control of the MTIO interface. Device allocation is done prior to accessing the device by the CL and is handled by the allocation routines in the ETC interface, with kernel support (this is largely independent of the magtape system).

All of the code listed above is part of the portable IRAF system (i.e., is OS independent and shared by all host versions of IRAF) until we get to the ZFIOMT driver. This is in the IRAF kernel and is host system dependent. At present the only version of ZFIOMT is the UNIX version; other versions, e.g., for VMS will follow as IRAF is updated to V2.10 on these systems. The UNIX version of ZFIOMT uses only generic UNIX ioctls and should compile on most UNIX systems without change. All of the system and device dependence has been concentrated in the tapecap file. The ioctls used to communicate with a UNIX tape device are fairly standard, but the semantics are often poorly defined and are certainly not standardized.

Beneath the IRAF driver are the host level magtape device drivers. A given host system may have more than one of these, typically one for each class of magtape device. Some systems, notably Sun with their ST (SCSI tape) driver, try to control more than one type of device with a single host driver. The host driver may come with the OS or may be supplied by a third party vendor.

Beneath the host driver is the actual tape device. Often these days this is a SCSI tape device such as a DAT or Exabyte. These devices are intelligent peripherals; control of the physical tape hardware resides in the tape unit. There is a small computer in each tape drive which communicates with the host computer via the SCSI interface, passing commands and data back and forth. The drive will buffer 256K to 512K of data passed in bursts over the SCSI bus, and then copied to or from the physical media at a much slower rate. These drives emulate variable size records by blocking and deblocking within the drive unit, and writing fixed size blocks to the media. Features such as error correction and data compression are also handled within the drive.

Although we usually speak of tape devices, the "magtape" device does not have to be a tape device at all. The IRAF magtape system can also make use of, e.g., a floppy disk, or anything that looks like a raw disk partition. The problem with the latter devices is that they usually don't support files and file positioning, hence can only be used to store one "file".

2.5.2.2. Driver features

A detailed description of the magtape driver is beyond the scope of this document. Briefly, the driver supports two main classes of devices, variable record size devices and fixed block devices. All file positioning is handled by the driver, and while the driver has the device open it is responsible for keeping track of the device position (the saved device position is passed in at open time and saved by the high level code at close time). A couple of dozen tapecap parameters are defined which describe the characteristics of each device, such as whether it supports variable records, the maximum record size, whether it can backspace, and so on. A socket or file based status logging feature is provided which allows detailed monitoring of the tape status during execution (see below).

In the simplest case the new magtape system, coupled with the UNIX version of the magtape driver, will emulate simple UNIX commands such as *tar* and *mt* insofar as the requests made to the host system and magtape device are concerned. That is, if one writes a series of files the only system requests made for each file will be open, write, and close. When reading a series of files in sequence the only requests made will be open, read, and close. Providing no file positioning is attempted it is possible to get by with no file positioning requests other than rewind. The driver provides extensive facilities for file positioning, using tapecap parameters to work around any shortcomings of the host system or device, but in case of trouble it is possible to operate with only open, close, read, and write requests, which should work for any device (unless it is truly broken).

2.5.3. Tapecap

The tapecap file, or magtape device capabilities file, defines the magtape devices known to the system and how to access these devices. While large portions of this file depend only upon the host operating system and device type and hence are fairly site independent, it will typically be necessary to customize the tapecap file to configure the IRAF magtape system for a site. In normal use the tapecap file is invisible to the user, but users with special problems may wish to learn more about tapecap to gain more control over the behavior of the magtape system.

2.5.3.1. Using tapecap

The standard tapecap file is the file `dev$tapecap`. A system environment variable `tapecap` is defined which by default points to this file.

Users can customize or manipulate tapecap entries in either of two ways. The user can have their own private copy of the tapecap file (much as is currently done with the termcap and graphcap files), by resetting the value of the `tapecap` environment variable to point to their local copy of the file. For example,

```
    cl> reset tapecap = home$tapecap
```

This may be necessary to customize a device entry, or add support for a local device not supported by the system wide tapecap file.

It is also possible to modify a tapecap device entry "on the fly", by overriding the values of specific tapecap parameters on the command line when the device is accessed. For example,

```
    cl> mtex "mta[:so=/dev/tty]"
```

would override the default value of the tapecap parameter "so" for device mta (in this case enabling status output logging and directing this output to the user terminal).

Specifying tapecap parameters on the command line is useful for experimentation but rapidly becomes tiresome if many commands are entered. In this case it becomes simpler to redefine `tapecap` to include the desired tapecap parameter overrides.

```
    cl> reset tapecap = ":so=/dev/tty"
```

As we see, the `tapecap` environment variable can be used to either specify the tapecap file name, or globally override specific tapecap parameters (all device entries are affected). The full form of the value string for `tapecap` is

`cl> reset tapecap =` [*tapecap-file*] [**`':'`** *capabilities-list*]

Any number of colon-delimited tapecap capabilities or parameters may be given.

It is beyond the scope of this document to detail all the tapecap parameters here. A reference paper for the magtape system is planned. For the present, there is a summary of the tapecap parameters in the ZFIOMT driver source file, `os$zfiomt.c`. For examples of actual usage refer to the tapecap file in the distributed system.

2.5.3.2. Configuring tapecap

The tapecap file uses the standard "termcap" file format, originally developed for BSD UNIX and adopted long ago for IRAF. Any UNIX system will probably have a manual page defining the termcap file format, if not this can be obtained from the IRAF group.

The distributed tapecap file is divided into three sections: the host machine specific device aliases (names like "mta" etc.), the site logical devices, and the generic device entries. To customize the tapecap file for a site you modify the first and second sections. To configure the tapecap file for a particular host machine you uncomment the entries for that host in the first section of the file. Sites which don't have a large network of hosts may prefer to combine the first two sections to simplify things. Site specific changes should never be made to the bottom, or generic, part of the tapecap file, as you will want to update this portion of the file when new versions of IRAF are released.

Don't be intimidated by the apparent complexity of some of the tapecap device entries. In most cases the generic device entry will already exist for the device you need to interface, and all you need to do is add a site entry which references the generic entry. In some cases the generic entry itself may be sufficient (for example, in the distributed SunOS version of tapecap, logical device "mtxb0" would provide access to Exabyte unit 0 interfaced with the Sun ST driver, "mtxb1" would be the same but unit 1, "mthp0" the HP 9 track tape on unit 0, and so on).

For example to interface Exabyte unit 2, using the Sun ST driver, as local device "mta", the following entry would suffice.

```
mta|        :tc=mtst2-exb:
```

If the generic device entry provided doesn't quite work and minor modifications are needed, these should be made to the *local* entry and not the standard generic entry. This is easily done with termcap parameter redefinitions. For example, in SunOS 4.1.2 (but not earlier versions of SunOS) there is a bug in the Sun ST driver which necessitates rewinding the tape after a tape scan is made to position to EOT (!). The capabilities ":se:nb" can be added to the tapecap entry for the device to workaround this bug, as in the following example.

```
mta|        :se:nb:tc=mtst2-exb:
```

The parameters mean that the device spaces past EOT in a read (se) and cannot backspace (nb). Neither of these things is actually true, but the effect is that the tape is rewound and spaced forward to get back to the desired file, working around the host level driver bug. Access will be less efficient than it should be, but the interface will work properly and the user does not have to be concerned with the problem.

As a final example, suppose we need to write a new tapecap entry from scratch because there is no generic entry for the device in the distributed tapecap file. To simplify things we assume that no special tapecap parameters are needed, i.e., that the standard UNIX defaults builtin to the driver will work for the device. We are upgrading from an old version of IRAF so we already have an old `dev$devices` file to work with. For the purposes of our example we use an old HP 88780 1/2 tape drive entry, but pretend that the device is something which is not already supported.

The old devices file entry was as follows.

```
mta         nrst20 nrst4 nrst12 nrst28 rst4 rst12 rst20 rst28
mta.6250    nrst20 nrst4 nrst12 nrst28 rst4 rst12 rst20 rst28
```

The minimal tapecap entry required to duplicate this is the following.

```
mta|mta6250|HP 88780 1/2 inch tape drive:\
        :al=nrst4 nrst12 nrst20 nrst28 rst4 rst12 rst20 rst28:\
        :dv=nrst20:lk=st4:tc=9tk-6250:
```

Here, the "al" parameter lists the device files to be allocated, the "dv" parameter is the device file to be used for i/o at the desired density (6250bpi in this case), and "lk" is the root file name to be used for the ".lok", or device status file. The "tc=" picks up the standard parameters for a 9 track 1/2 inch tape drive operating at 6250 bpi. Two device aliases are defined, "mta" and "mta6250".

2.5.3.3. Supported devices

The IRAF magtape system itself should support almost any magtape device if properly configured. What devices are actually supported at a site depends upon the tapecap file, and in particular upon the host system (different host systems have different tapecap files).

Device	*Driver*
QIC-11 cartridge tape	Sun st
QIC-24 cartridge tape	Sun st
QIC-150 cartridge tape	Sun st
Pertec compatible 1/2 inch drives	Xylogics
HP 88780 1/2 inch drive	Sun st

WangDAT 1300, 2000	ApUNIX
HP DAT	ApUNIX
Exabyte 8200, 8500	ApUNIX, Sun st, Ciprico
DC2000 cartridge tape	A/UX tc
FDHD floppy disk	A/UX fd

As an example, the tapecap file distributed in the V2.10.0 release of Sun/IRAF supported the devices listed in the table above. New devices are constantly being added. As V2.10 IRAF propagates to the various platforms on which IRAF is supported, similar tapecap files will be configured for those systems.

2.5.4. Status output

The new magtape driver has a facility known as status output logging which can be used to monitor interactively lengthy tape jobs while i/o is in progress. The status output facility can also be useful for debugging magtape problems.

In its simplest form, the status output may be directed to a file, e.g., an actual text file, or a terminal device. Status output is enabled by setting the "so" option in tapecap. For example,

```
cl> mtex "mta[:so=/tmp/mta.out]"
```

would enable status output logging and direct the output text to the file /tmp/mta.out. Likewise,

```
cl> mtex "mta[:so=/dev/ttyp7]"
```

would enable status output and direct the output to a pseudoterminal, e.g., an xterm window. When this form of status output logging is used one sees the raw, driver level status messages, as in the following example.

```
cl> mtex "mta[:so=/dev/tty]"
open device tc2n\n
devtype = Apple Tape 40SC
tapetype = DC2000
tapesize = 38500
density = na
blksize = 8192
acmode = read
file = -1
record = -1
nfiles = 0
tapeused = 0
device tc2n opened on descriptor 4
rewinding...
done\n
position to file 1\n
file = 1
record = 1
reading...\n
recsize = 65536
record = 9
tapeused = 64
   (etc.)
```

The UNIX version of the new magtape driver also has an option to direct status output to a TCP/IP socket, which can be anywhere on the network. For this option to be useful one must have a program which will listen on the designated socket, accept the connection when the magtape driver tries to open a connection, and then read and process the status messages (which are still text, exactly as in the example above).

Status output to a socket is enabled by giving a host name instead of a file name in the "so" directive:

```
        cl> mtex "mta[3:so=lepus]"
```

to examine file 3, directing status messages to a socket on node "lepus".

An X11 client application called *xtapemon* has been developed to listen on a socket, read messages from the tape driver, and provide a real-time display of the status of the tape device. While not included in the V2.10 IRAF distribution, this utility will be available later in 1992 as part of the X11 support package currently under development.

2.5.5. Error recovery

Considerable effort went into "bullet proofing" the new magtape system to make it recover gracefully from ctrl/c interrupts or other program aborts. In practice it can be very difficult to reliably catch and recover from interrupts in a multiprocess, distributed system such as IRAF. No doubt there are still conditions under which an interrupt will leave the magtape system in a bad state, but in most circumstances the system should now be able to successfully recover gracefully from an interrupt.

If it is necessary to interrupt a tape operation, it is important to understand that the host system will not deliver the interrupt signal to the IRAF process until any pending i/o operation or other driver request completes. For example, in a read operation the interrupt will not be acted upon until the read operation completes, or in a tape positioning operation such as a rewind or file skip forward, the interrupt will not be acted upon until the tape gets to the requested position. For a device such as a disk you rarely notice the delay to complete a driver request, but for a magtape device these operations will take anywhere from a few seconds to a few tens of seconds to complete. Type ctrl/c once, and wait until the operation completes and the system responds.

If a magtape operation is interrupted, the IRAF error recovery code will mark the tape position as undefined (*devstatus* will report a file number of -1). This will automatically cause a rewind and space forward the next time the tape is accessed. The rewind is necessary to return the tape to a known position.

2.5.6. Device optimization

In addition to making it possible to characterize the behavior of a magtape device to permit the device to be accessed reliably, the tapecap facility is used to optimize i/o to the device. The parameter "fb" may be specified for a device to define the "optimum" FITS blocking factor for the device. Unless the user explicitly specifies the blocking factor, this is the value that the V2.10 *wfits* task will use when writing FITS files to a tape. Note that for cartridge devices a FITS blocking factor of 22 is used for some devices; at first this may seem non-standard FITS, but it is perfectly legal, since for a fixed block size device the FITS blocking factor serves only to determine how the program buffers the data (for a fixed block device you get exactly the same tape regardless of the logical blocking factor). For non-FITS device access the magtape system defines an optimum record size which is used to do things like buffer data for cartridge tape devices to allow streaming.

Some devices, i.e., some DAT and Exabyte devices, are slow to switch between read and skip mode, and for files smaller than a certain size, when skipping forward to the next file, it will be faster to read the remainder of the file than to close the file and do a file skip forward. The "fe" parameter is provided for such devices, to define the "file equivalent" in kilobytes of file data, which can be read in the time that it takes to complete a short file positioning operation and resume reading. Use of this device parameter in a tape scanning application such as *rfits* can make a factor of 5-10 difference in the time required to execute a tape scan of a tape containing many small files.

On most cartridge tape devices backspacing, if permitted at all, is very slow. On such devices it may be best to set the "nf" parameter to tell the driver to rewind and space forward when backspacing to a file.

2.5.7. MTIO interface changes

A number of new routines were added to the MTIO (magtape i/o) programming interface. These are documented in the summary of programming environment revisions given below. Existing magtape applications may benefit from being modified to make use of these new routines.

2.6. Graphics and Imaging Subsystem Enhancements

2.6.1. New graphics applications

New tasks for histogram plotting, radial profile plotting, and producing lists of the available graphics devices have been added to the PLOT package. All of the tasks in this package were revised to add support for world coordinates. A related task for drawing world coordinate grid overlays on images or plots was added to the IMAGES.TV package. See the appropriate sections of *IRAF and NOAO package revisions* below for further information on these changes.

2.6.2. Graphics system changes

2.6.2.1. Encapsulated postscript SGI kernel

A new encapsulated postscript SGI kernel has been added. Graphics output may be directed to any of the logical graphics devices *eps*, *epsl*, *epshalf*, etc. to render a plot in encapsulated postscript, e.g., for inclusion as a figure in a document. For example,

```
    cl> prow dev$pix 101 dev=eps; gflush
```

will leave a ".eps" file containing the plot in the current directory. The command "gdev dev=eps" will produce a list of the available EPS logical graphics devices.

2.6.2.2. Graphics output to the default printer

Graphics output (SGI postscript) can now be directed to the UNIX default printer device by directing output to any of the logical devices "lpr", "lp", or "lw".

```
    cl> prow dev$pix 101 dev=lpr
```

Output will be sent to the default device for the UNIX *lpr* command (set by defining "PRINTER" in your UNIX environment). This makes it possible to make immediate use the distributed IRAF graphcap without having to add entries for specific local devices (although you may still wish to do so).

2.6.2.3. Tick spacing algorithm improved

The algorithm used to draw the minor ticks on IRAF plots was replaced by an improved algorithm contributed by the STScI STSDAS group (Jonathan Eisenhamer in particular). This was derived from similar code in Mongo.

2.6.2.4. Graphics metacode buffer

The default maximum size of the graphics metacode buffer in the CL, used to buffer graphics output for cursor mode interaction, was increased from 64K to 128K graphics metacode words (256Kb). The `cmbuflen` environment variable may be used to change this value.

2.6.2.5. IMTOOL changes

The *imtool* display server (SunView) was enhanced to add additional builtin color tables, support for user color tables, and setup panel control over the screen capture facilities. A version supporting encapsulated postscript output is in testing. This will be the final version of the

SunView based version of imtool (the new display servers are all X window system based).

2.6.2.6. IMTOOLRC changes

The `imtoolrc` file, used by display servers such as *imtool* and *saoimage* to define the available image frame buffer configurations, has been relocated to the DEV directory to make it easier for local site managers to customize. The IRAF install script now uses a link to point to this file, rather than copying it to /usr/local/lib. The maximum number of frame buffer configurations was increased from 64 to 128.

2.6.2.7. X11 support

The released version of V2.10 does not contain any changes insofar as X11 support is concerned. Since our X11 support code is host level stuff, with minimal ties to IRAF per se, it is being developed independently of the V2.10 release and will be distributed and installed as a separate product.

2.7. Image Structures

2.7.1. Image I/O (IMIO)

The image i/o interface (IMIO) is that part of the IRAF system responsible for imagefile access and management. The changes to IMIO for V2.10 included the following.

2.7.1.1. Null images

Null images are now supported for image output, much like the null files long supported by the file i/o system. For example,

```
    cl> imcopy pix dev$null
```

would copy the image "pix" to the null image. This exercises the software but produces no actual output image. Unlike null files, null images do not work for input since images have some minimal required structure, whereas files can be zero length.

2.7.1.2. Preallocating pixel file storage

In the UNIX versions of IRAF, when a new image is created storage is not physically allocated for the output image until the data is written. This is because most UNIX systems do not provide any means to preallocate file system storage. The UNIX/IRAF file creation primitive *zfaloc*, used by IMIO to create pixel files, now provides an option which can be used to force storage to be physically allocated at file creation time. This feature is enabled by defining the environment variable `ZFALOC` in the UNIX environment. For example,

```
    % setenv ZFALOC
```

can be entered before starting IRAF to cause space for all pixel files to be preallocated as images are created. If it is not desired to preallocate all image storage (there is a significant run-time overhead associated with preallocating large images) then a value string can be given to specify which types of images to preallocate storage for. For example,

```
    % setenv ZFALOC /scr5
```

would preallocate only those pixel files stored on the /scr5 disk, and

```
    % setenv ZFALOC "/scr5,zero"
```

would preallocate only images on /scr5, or images containing the substring "zero" in the image name. In general, the string value of `ZFALOC` may be the null string, which matches all images, or a list of simple pattern substrings identifying the images to be matched.

In most cases the default behavior of the system, which is to not physically allocate storage until the data is written, is preferable since it is more efficient. The preallocation option is provided for users who, for example might have an application which spends a lot of time computing an image, and they want to ensure that the disk space will be available to finish writing out the image.

2.7.1.3. Image templates now sorted

As mentioned earlier in the *System Management* section, UNIX/IRAF now sorts UNIX directories. One result of this is that the sections of image templates which are expanded via pattern matching against a directory will now be sorted, or at least logically ordered (the final output list will not necessarily be fully sorted if string substitution is being performed - this is the reason the output list itself is not sorted).

2.7.1.4. The dev$pix test image

Minor changes were made to clean up the image header of the standard test image `dev$pix`. A second test image `dev$wpix` has been added. This is the same image, but with a different header containing a test world coordinate system (actually the image is just a second image header pointing to the `dev$pix` pixel file, now shared by both images).

2.7.2. Image kernels (IKI)

The IMIO image kernels are the data format specific part of the IRAF image i/o subsystem. Each kernel supports a different image format. Existing disk image formats range from the conventional image raster formats (OIF and STF) to the photon image format (QPF and QPOE) and the pixel mask image format (PLF and PMIO/PLIO). There also exist special image kernels which allow IMIO to be used to access non-disk storage devices such as image display frame buffers.

2.7.2.1. New PLF image kernel

A new image kernel, the PLF image kernel, has been added which allows IRAF PMIO/PLIO pixel masks to be stored as images. This kernel first appeared as a patch to version 2.9 IRAF but was actually developed within V2.10.

Pixel mask images may be created, deleted, read, written, etc. like other IRAF images, but the image is stored in a special compressed format designed specially for image masks. An image mask is an N-dimensional raster-like object wherein typically there are regions of constant value but arbitrary shape. Masks are used by applications involving image decomposition. The image is decomposed into regions of different types, the type of region being very dependent upon the type of image analysis being performed. A special type of mask image is the bad pixel mask, used to flag the bad pixels in an image. Other applications use masks for data quality, to identify the region or regions to be used for analysis, and so on.

A PMIO image mask is a special case of a PLIO pixel list. Masks can exist and be accessed independently of the image i/o system, but the PLF image kernel allows a mask to be stored and accessed as an IRAF image. Any IRAF application which operates upon images can operated upon a mask image. For example, the *imcalc* (image calculator) program in the SAO XRAY package can be used to combine images and masks, or compute new masks by evaluating an algebraic expression over an image.

Mask images have the reserved image extension ".pl". Some of the features of mask images are illustrated by the following example.

```
cl> imcopy dev$pix pix.pl
dev$pix -> pix.pl
cl> imstat dev$pix,pix.pl
#     IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
    dev$pix    262144     108.3     131.3       -1.    19936.
     pix.pl    262144     108.3     131.3        6.    19936.
```

This is a sort of worst case test of the mask encoding algorithm, since our test image is not a mask but a noisy image (and hence not very compressible). Note that masks must be positive valued, hence the MIN value is different for the two images. Storing `dev$pix` as a mask does not result in any significant compression, but for a real mask very high compression factors are possible. The compression algorithm used in PLIO is simple and fast, combining 2D run length encoding and a differencing technique in a data structure allowing random access of multidimensional masks (masks are not limited to one or two dimensions).

For further information on pixel lists and pixel masks, see the document `plio$PLIO.hlp` in the online system. This is also available as `plio.txt.Z` in the IRAF network archive.

2.7.2.2. OIF image kernel changes

The OIF image kernel is the kernel for the old IRAF image format - this is still the default IRAF image format. Revisions to the OIF kernel for V2.10 included the following items.

- A valid image header is now written even if an application which writes to a new image is aborted. Previously, the pixel file would be written but the image header would be zero length until the image was closed. If an image creation task aborts the image will likely be incomplete in some way, e.g., part of the pixels may not have been written to, or the header may be missing application specific keywords. By valid image header we mean that the header will be valid to IMIO, allowing the image to be accessed to try to recover the data, or to delete the image.
- The image header file of a new image is now written to and closed at image create time, then reopened at image close time to update the header. This frees one file descriptor, an important consideration for applications which for some reason need to write to dozens of output images simultaneously.
- The OIF image kernel uses file protection to prevent inadvertent deletion of the image header file. In UNIX/IRAF systems file delete protection is simulated by creating a ".." prefixed hard link to the file. This could cause problems with images if the user deleted the image header file outside of IRAF, leaving the ".." prefixed link to the file behind. A subsequent image create operation with the same image name would fail due to the existence of the hidden ".." prefixed file. It is unlikely that such a file (with a ".." prefix and a ".imh" extension) could ever be anything but an old IRAF image header file, so the system will now silently replace such files when creating a new image.

A couple of related system changes were made which, while not implemented in the OIF kernel, do involve the OIF or ".imh" image format. The default template `login.cl` now defines the environment variable `imtype` and sets it to "imh". The environment variable `min_lenuserarea` is also defined now at login time, with a default value of 20000, allowing image headers with up to about 250 header parameters.

2.7.2.3. STF image kernel changes

The STF image kernel is the kernel for the online HST (Hubble Space Telescope) image format. This format allows multiple images to be grouped together in a single "group format" image. There is a common image header stored in a FITS-like format, as well as a small fixed format binary header associated with each image in the group.

- A check was added for a group index out of range. This yields a more meaningful error message about no such group, rather than having IMIO complain about a reference out of bounds on the pixel file.
- Support was added for non-group STF images (GROUPS=F in the header). At first glance a non-group group format image might seem a little silly, but it turns out that a non-group STF image with a zero length group parameter block is very close to "FITS on disk", since the header is FITS-like and the image matrix is simple. It is still not true FITS since the header and pixels are stored in separate files and the pixels are not encoded in a machine independent form, but on UNIX hosts which are IEEE standard and not byte swapped, it comes close enough to be useful for communicating with external applications in some cases.

A true FITS image kernel is planned for IRAF. This will probably appear in one of the V2.10 patches, sometime during 1992.

2.7.2.4. QPF image kernel changes

The QPF image kernel is the interface between the QPOE (position ordered event file) interface and IMIO, allowing QPOE event files to be accessed as images by general IRAF applications. Photon "images" are unique in that the image raster is generated at runtime as the image is accessed, optionally passing the photon list through event attribute and spatial filters, and sampling the photons to produce a raster image. For example,

```
cl> imcopy "snr[time=@snr.tf,bl=4]" snr.imh
```

would filter the event list stored in the file "snr.qp" by the time filter stored in file "snr.tf", sample the image space blocking by a factor of 4, and store the resultant image raster in the OIF image file "snr.imh".

```
cl> display "snr[time=@snr.tf,bl=4]" 1
```

would display the same image raster without creating an intermediate disk image.

The changes to the QPF interface for V2.10 included the following.

- A bug was fixed which would prevent very long filter expressions from being correctly recorded in the IMIO image header. The current version of IMIO stores applications header parameters in FITS format, hence multiple FITS cards are required to store long filter expressions. The bug would cause only one such card to be output.
- A new facility was added which allows general expressions to be computed for the input event list and stored as keywords in the output image header. The header of the input QPOE file can contain one or more parameters named *defattr1*, *defattr2*, and so on. If present, the string value of each such parameter should be a statement such as

  ```
  exptime = integral time:d
  ```

  which will cause a keyword named "exptime" to be added to the output image header, the scalar value of the keyword being the value of the expression on the right. Currently, only the integral function is provided. This computes the included area of a range list field of the event attribute expression, such as a time filter. In the example, "time" is the name of the event attribute to be integrated, and the ":d" means use a range list of type double for the computation. The data types currently supported are integer, real and double.

Other minor changes to QPF included improvements to the error recovery, and other changes to support very large filters.

2.7.3. World coordinate system support (MWCS)

MWCS is the IRAF world coordinate system package, one of the major new system interfaces developed for the new image structures project. A full description of the MWCS interface is given in the file `mwcs$MWCS.hlp` in the online system, and in the file `mwcs.txt.Z` in the IRAF network archive.

2.7.3.1. Applications support

MWCS was first released in V2.9 IRAF and at the time the interface was new and few applications were yet using it. In V2.10 IRAF most (but not all) applications which deal with coordinates now use MWCS. These include all of the system plotting tasks, and the spectral reduction packages. Details of the MWCS support added to the system and science applications in V2.10 are given in the *IRAF and NOAO package revisions* section of this revisions summary.

2.7.3.2. New function drivers

Function drivers for the *arc*, *sin*, and *gls* nonlinear sky projections were added, as well as a special function driver for the *multispec* image format. The latter allows general programs which don't know anything about spectra to access and display spectra in world coordinates, e.g., for plotting.

2.7.3.3. WCS attributes

The maximum number of "attributes" per WCS was increased from 64 to 256, mainly to support the multispec function driver, which makes heavy use of attributes. A limit on the size of a single attribute value string was removed, allowing arbitrarily large attribute values to be stored.

2.7.3.4. Axis mapping

Axis mapping is now fully implemented. Axis mapping is used when, for example, you extract a 2 dimensional section from a 3 dimensional image and write the result to a new image. Axis mapping allows the 2 dimensions of the new image to be mapped backed into the original 3 dimensional WCS, making it possible to get the full physical coordinates (which are 3 dimensional) for any point in the extracted image. A new header keyword `WAXMAP`*nn* was added to store the axis map in image headers.

2.7.3.5. MWCS save format

The newer IRAF image formats such as QPOE are capable of storing arbitrarily complex objects such as a MWCS in an image header keyword. In the case of a stored MWCS object, the MWCS interface is responsible for encoding the object in its external storage format, and restoring it to a MWCS descriptor when the MWCS is accessed. The code which does this was revised to add a new version number for the stored V2.10 MWCS, to make it possible to differentiate between the MWCS save formats used in V2.9 and V2.10 and allow recovery of MWCS objects from data files written under V2.9.

2.7.3.6. Bug fixes

Since MWCS is a new interface and V2.10 saw its first real use in applications, a number of interface bugs were discovered and fixed. Most of the bugs turned out to be in the part of MWCS which converts between the internal MWCS WCS representation, and the FITS WCS representation, used for those image formats that still use FITS-like headers. Bug fixes included a problem with the treatment of CROTA2, a problem with the FITS CD matrix, an axis mapping problem that caused "dimension mismatch" errors, and a problem with support for the old-style CDELT/CROTA which could result in a singular matrix during WCS inversion when compiling a transformation.

2.7.4. Event files (QPOE)

The QPOE interface is the interface responsible for creating and accessing *event files*, the main data format produced by event counting detectors. We summarize here the main enhancements to QPOE made during the preparation of V2.10. Some of these features appeared earlier in the patches made to V2.9 IRAF but these revisions have never been formally documented so we summarize both the old and new revisions here. See also the discussion of the QPF image kernel given earlier.

2.7.4.1. Blocking factors

The builtin default blocking factor, when sampling the event list to make an image raster, is one. This may be changed on a per-datafile basis by defining the parameter *defblock* in the QPOE file header.

2.7.4.2. Parameter defaults

Since parameters such as the blocking factor can be set at a number of levels, a parameter defaulting scheme was defined to determine the precedence of parameter overrides. The lowest level of precedence is the builtin default. Next is any datafile specific value such as *defblock*. A value such as *blockfactor* set in the QPDEFS file will override the datafile default value if any. Specifying the parameter value in a runtime filter expression or *qpseti* call is the highest order of precedence and will override any other setting.

Another way to think of this is the more recently the parameter was set, the higher the precedence. The oldest value is the default builtin to the code. Next is the datafile value, usually set when the datafile was created. Next is the QPDEFS macro file, usually set by the user or for a site. Last (highest precedence) is the value set by the user when the data is accessed.

2.7.4.3. Referencing the current datafile in macros

A special symbol `$DFN` is now recognized during macro expansion and if present is replaced by the filename of the current QPOE file. This allows macros to be written which automatically perform some operation involving the datafile to which they applied, e.g., computing an attribute list from aspect or data quality information stored in the datafile header.

2.7.4.4. Bitmask expressions

Bitmask expressions may now be negated, e.g., "%3B" is the mask 03 octal, "!%3B" or "!(%3B)" is the complement of 03 octal.

2.7.4.5. Large time filters

A number of changes and bug fixes were made to QPOE for V2.10 to support large time filters. These are lists of "good time" intervals used to filter the event list. These large time filters may contain several hundred double precision time intervals spanning the exposure period. Often a large time filter is combined with a complex spatial filter (PLIO mask) to filter an event list consisting of from several hundred thousand to several million events. QPOE can handle this efficiently regardless of whether the data is temporarily or spatially sorted and regardless of the complexity of the temporal or spatial filters.

A number of minor bugs were fixed caused by the large filter expressions overflowing various buffers. The default sizes of the program and data buffers used to compile filter expressions were increased to allow all but very large filters to be compiled without having to change the defaults. If the buffers overflow more space can be allocated by setting the `progbuflen` and `databuflen` parameters in QPDEFS (these buffers are not dynamically resized at runtime for efficiency reasons). During testing with large time filters it was discovered that a routine used to test floating point equality was being used inefficiently, and compilation of large time filters was taking a surprisingly long time. A change was made which reduced the time to compile large filters by a factor of 10 to 15.

2.7.4.6. Default filters

QPOE now fully supports default event attribute and spatial filtering of data at runtime. The idea is that the full data set (all events) is stored in the QPOE file, along with default event attribute and spatial filters. When the data is accessed these filters are, by default, automatically applied. Any user specified event attribute or spatial filters are "added" to the builtin default filters to produce the combined filter used when the event list is accessed. The point of all this is to make it easy for the user to modify or replace the default filters and "reprocess" the raw event list. In effect the raw and processed event list are available in the same file.

The default filter and mask, if any, are stored in the datafile header parameters `deffilt` and `defmask`. If the datafile contains multiple event lists a default filter or mask may be associated with each list by adding a period and the name of the event list parameter, e.g., "deffilt.foo" would be the default filter for the event list "foo".

By default, if a default filter or mask exists for an event list it is automatically applied when the event list is accessed. Use of the default filter or mask can be disabled in QPDEFS with either of the following statements:

```
    set nodeffilt
    set nodefmask
```

The default filter and mask can also be disabled in a filter expression by adding a "!" before the expression, as in the following example.

```
    display "snr[!time=@times.lst,pha=(3,8:11)]"
```

There are actually several variants on the "=" assignment operator used in filter expressions such as that above. The operator "=" is equivalent to "+=", meaning the expression term on the right adds to any existing filter specified for the event attribute on the left. The operator ":=" on the other hand, means *replace* any existing filter by the new filter expression.

2.7.4.7. Alternate coordinate systems

The event structure used to represent each event in an event list may contain multiple coordinate systems if desired, for example, detector and sky coordinates. One coordinate system should be defined as the default when the event list is created, and if the event list is to be indexed it must be sorted using the coordinate system specified as the default. Any coordinate system may be used when the data is accessed however; this can result in quite different views of the same data set. For example,

```
    cl> display snr.qp 1
```

would display the QPOE file "snr.qp" in display frame 1, using all defaults for the event list, blocking factor, filter, mask, and so on. The default event coordinate system would probably be sky coordinates. To display the same event list in detector coordinates, one could enter the following command.

```
    cl> display "snr.qp[key=(dx,dy)]" 1
```

This assumes that the event structure fields "dx" and "dy" are defined somewhere, either in the datafile header or in QPDEFS (otherwise the actual field datatype-offset codes must be given).

Currently if the QPOE datafile has a WCS associated with it this WCS can only refer to one coordinate system, normally the default event coordinate system. Additional WCS can be stored in the QPOE file, either in the stored MWCS object or as separate MWCS objects, but at present there is no mechanism for selecting which will be used (if the parameter `qpwcs` exists in the QPOE file header, this is assumed to be a stored MWCS object and this is the WCS which will be used).

2.8. System Specific Changes

2.8.1. Sun/IRAF

Since V2.10 has only just been completed and at this stage has only been released on Sun platforms, there are few system specific revisions to report. For SunOS there were several system specific revisions worth noting here.

- The HSI binaries for the sparc architecture are now statically linked. This makes them considerably larger, but avoids problems with SunOS complaining about shared library version mismatches (note that we refer here to to Sun shared libraries - this is not a problem with the IRAF shared library facility since we control the version numbers).
- The HSI binaries for the Motorola architectures (f68881 and ffpa) are *not* statically linked since they cannot be without running into linker problems warning about f68881_used etc. To avoid or at least minimize warnings about SunOS shared library versions the system was linked on 4.1.1 instead of 4.1.2. SunOS 4.0.3 versions of the Motorola HSI binaries are also available upon request.
- The system as distributed was linked with the name server library, `-lresolv`. This means that if the local host is configured to use the name server, IRAF will be able to access almost any host on the Internet without an entry in the `/etc/hosts` file on the local host.

Additional system specific changes will be reported in the documentation distributed with each release, as V2.10 is moved to the various platforms for which IRAF is supported.

3. IRAF and NOAO package revisions

The revisions for the various packages in the IRAF core system and in the NOAO packages are summarized below. There have been many changes with this release. Users are encouraged to refer to the help pages, user's guides provided with some packages, revisions notes, and more recent issues of IRAF Newsletters for more details. Comprehensive notes on systems and applications modifications are in the files `pkgnotes.v210.Z` and `sysnotes.v210.Z` in the directory `iraf/v210` in the network archive. This summary can be read online by typing *news*. Revisions notes for the various packages can also be accessed online as in the following example.

```
cl> phelp dataio.revisions opt=sys
```

One of the system changes that affects many tasks in the IMAGES, PLOT, and LISTS packages as well as most tasks in the spectroscopic packages in NOAO is the full implementation of the world coordinate system (WCS), introduced in V2.9 but not fully integrated into the IRAF and NOAO tasks at that time. There are currently three classes of coordinate systems: the logical system is in pixel units of the current image section; the physical system is in pixel units of the parent image (for a WCS associated with a raster image); the world system can be any system defined by the application, e.g. RA and DEC or wavelength. In general, this should be transparent to the user since most defaults have been chosen carefully so that tasks perform the same as in V2.9. The NOAO spectroscopic tasks also use the WCS extensively, but again, this should be transparent to the user, although it can cause some problems with backward compatibility. Users will notice the biggest difference in the image headers with additional keywords added by the interface. Two tasks in the PROTO package may help the interested user understand more about the WCS - see *wcsedit* and *wcsreset*. Technical notes on the WCS are available in a paper in the `iraf$sys/mwcs` directory. Type the following to read more about the MWCS interface.

```
cl> phelp mwcs$MWCS.hlp fi+
```

3.1. Changes to the IRAF system packages

3.1.1. DATAIO package modifications

- The output defaults for *wfits* have been modified. If the pixel type on disk is real or double precision the output will be IEEE format (FITS bitpix = -32 or -64, respectively). If the pixel type on disk is short integer then the output will be integer format (bitpix = 16) as before. These defaults can of course be changed by modifying the parameters for *wfits*.
- The *wfits* "blocking_factor" parameter can accept values up to and including 10 for variable blocked devices, i.e. 1/2" tape drives, Exabytes, and DATs. If the "blocking_factor" parameter is set to "0", as in the default, the value is read from the "fb" parameter in the tapecap file (usually 10 for variable blocked devices). For fixed blocked devices such as cartridge tape drives the default value for the "fb" parameter in the tapecap file is 22 (used to determine a buffer size) and the block size of the device is given by the "bs" parameter.
- All tasks were modified to accept tapecap parameters as part of the magtape file name syntax. Tapecap parameters can now be appended to the magtape file name to add to or override values in the tapecap file. For example, `"mta6250[:se]"` is a valid magtape file name (the "" are important when tapecap parameters are specified at execution time). See the file `os$zfiomt.c` for more details about the tapecap parameters.
- The *rfits* task has been modified to permit a short last record, i.e. a last record that has not been padded out to 2880 bytes. No warning messages are issued.
- *rfits* was modified to map ASCII control characters in the header to blanks.

- The sequence number appended to disk file names by *rfits* and *wfits* was modified to 4 digits, i.e. 0001 - 9999.

3.1.2. IMAGES package modifications

- WCS (world coordinate system) support was added to all tasks in the IMAGES package that could introduce a coordinate transformation. Some tasks had already been modified for the V2.9 release. These tasks (*blkavg*, *blkrep*, *geotran*, *imcopy*, *imlintran*, *imshift*, *imslice*, *imstack*, *imtranspose*, *magnify*, *register*, *rotate*, and *shiftlines*), upon execution, will update the image header to reflect any transformation.
- The *listpixels* task was modified to list the pixel coordinates in either logical, physical, or world coordinates. A format parameter was added to the task for formatting the output pixel coordinates taking precedence over the WCS in the image header, if used.
- A new *imcombine* task was added to the package. This new task supports image masks and offsets, and has an assortment of new combining algorithms. See the help pages for complete details on this powerful new task.
- The *imhistogram* task has a new "binwidth" parameter for setting histogram resolution in intensity units.
- The default values for the parameters "xscale" and "yscale" in the *register* and *geotran* tasks were changed from INDEF to 1.0, to preserve the scale of the reference image.

3.1.3. IMAGES.TV package reorganization and modifications

- The TV package has been reorganized. The IIS dependent tasks have been moved into a subpackage, TV.IIS. The *imedit*, *imexamine*, and *tvmark* tasks that were previously in PROTO have been moved to TV.
- A new task *wcslab* developed at STScI by Jonathan Eisenhamer was added to the package. *wcslab* overlays a labeled world coordinate grid on an image using WCS information stored in the header of the image or in parameters of the task.
- *imexamine* was modified to use WCS information in axis labels and coordinate readback, if selected by the user. Two new parameters, "xformat" and "yformat", were added to specify the format of the WCS if it is not present in the header of the image.
- A new option was added to *imexamine* to allow for fitting 1D gaussians to lines or columns.
- *tvmark* had two cursor key changes. The "d" keystroke command that marked an object with a dot was changed to "."; the "u" keystroke command that deleted a point was changed to "d".

3.1.4. LISTS package modifications

- Two new parameters were added to the *rimcursor* task, "wxformat" and "wyformat". These parameters allow the user to specify the coordinate formats for the output of the task and override any values stored in the WCS in the image header.

3.1.5. OBSOLETE package added

- An new package called OBSOLETE was added. Tasks will be moved to this package as they are replaced by newer tasks in the system. OBSOLETE tasks will then be removed at the time of the next release.
- *mkhistogram*, *imtitle*, *radplt*, and the old *imcombine* task have been moved to the OBSOLETE package. Users should become familiar with *phistogram*, *hedit*, *pradprof*, and the new *imcombine* and use them instead since *mkhistogram*, *imtitle*, *radplt*, and *oimcombine* will be retired with the next release.

3.1.6. PLOT package modifications

- A new task called *phistogram* was added to the PLOT package. This task takes input from an image or from a list and allows full control over the plotting parameters. This task replaces the *obsolete.mkhistogram* task.
- A new task *pradprof* was added to the PLOT package. This task plots or lists the radial profile of a stellar object. This task replaces the *obsolete.radplt* task.
- A new task called *gdevices* was added to the package. *gdevices* prints available information known about a particular class of device. The default parameters are set up to print a list of the available stdimage devices in the standard graphcap file.
- WCS support was added to the tasks *graph*, *pcol(s)*, and *prow(s)*. A new parameter called "wcs" was added to define the coordinate system to be used on the axis, either logical, physical or world. Two additional parameters, "xformat" and "yformat", were also added to allow the user to set the format for axis labels overriding any values in the image header. The "xlabel" parameter values were extended to include the special word "wcslabel" to select the WCS label for the x axis from the image header.
- WCS support was added to the task *implot*. A new parameter called "wcs" was added to define the coordinate system for plotting, either logical, physical, or world. Two new ":" commands were also added: ":w" allows the user to set a new WCS type interactively; ":f" allows the user to change the axis format, for instance, ":f %H" would change the axis to read h:m:s, if the world coordinate RA had been defined in the image header. The "space" key now prints out the coordinate and pixel value information.
- *graph* has a another new parameter "overplot" that allows the user to overplot multiple plots with different axes.
- The default parameters for "floor" and "ceiling" in *contour* and *surface* were changed to INDEF.

3.1.7. PROTO package reorganization and modifications

This package has been reorganized. The PROTO package now resides in the core system. Tasks in the old PROTO package that were general image processing tools were kept in this new PROTO package. Tasks that were more data reduction specific were moved to the new NPROTO package in the NOAO package. The current PROTO package now contains the tasks *binfil*, *bscale*, *epix*, *fields*, *fixpix*, *hfix*, *imalign*, *imcentroid*, *incntr*, *imfunction*, *imreplace*, *imscale*, *interp*, *irafil*, *joinlines*, *suntoiraf*, *wcsedit*, and *wcsreset*.

- The new task *hfix* was added to the package. This task allows the user to extract the image headers into a text file, view or modify this file with a specified command, and then update the image header with the modified file.
- A new task called *suntoiraf* was added to this package. This is a host dependent task that will most likely be useful only on Sun's. This task converts a Sun raster file into an IRAF image.
- Two new tasks dealing with the WCS were added to this package, *wcsreset* and *wcsedit*. *wcsreset* resets the coordinate systems in the image header to the logical coordinate system. *wcsedit* allows the user to modify the existing WCS or to create a new WCS and then update the image header.
- A new version of *bscale* was added to the package. The task now takes a list of input images and output images.
- A new version of *imfunction* was added to the package. The task supports many more functions, for example log10, alog10, ln, aln, sqrt, square, cbrt, cube, abs, neg, cos, sin, tan, acos, asin, atan, hcos, hsin, htan, and reciprocal.

- *imreplace* was modified to support the "ushort" pixel type.
- The task *join* has been renamed *joinlines*.

3.1.8. Additions to XTOOLS and MATH

Programmers may be interested in the following new additions to the XTOOLS and MATH libraries.

- The interactive non-linear least squares fitting package INLFIT, developed by Pedro Gigoux at CTIO, was added to XTOOLS.
- The 1D image interpolation routines in the MATH library were modified to support sinc interpolation.

3.1.9. Glossary of new tasks in the IRAF core system

Task		Package		Description
gdevices	-	plot	-	List available imaging or other graphics devices
hfix	-	proto	-	Fix image headers with a user specified command
imcombine	-	images	-	Combine images pixel-by-pixel using various algorithms
phistogram	-	plot	-	Plot or print the histogram of an image or list
pradprof	-	plot	-	Plot or list the radial profile of a stellar object
suntoiraf	-	proto	-	Convert Sun rasters into IRAF images
wcsedit	-	proto	-	Edit the image coordinate system
wcslab	-	tv	-	Overlay a displayed image with a world coordinate grid
wcsreset	-	proto	-	Reset the image coordinate system

3.2. Changes to the NOAO Packages

An updated version of the *observatory* task has been installed at the NOAO level. The task sets observatory parameters from a database of observatories or from the parameters in the task itself. Many packages and tasks within the NOAO packages that need information about where the data was observed use information supplied by the *observatory* task.

3.2.1. ARTDATA package modifications

A new version of the ARTDATA package was released with IRAF version 2.9.1. A summary of those changes plus modifications since that update are listed below. A more comprehensive list of the changes included in V2.9.1 are discussed in IRAF Newsletter Number 10.

- A new task *mkechelle* has been added that creates artificial 2D echelle images.
- A new task *mkexamples* has been added. The task is intended to generate a variety of artificial images to be used in testing or demonstrations.
- A new task *mkheader* adds or modifies image headers using a header keyword data file.
- The *mk1dspec* task was modified to create multispec and echelle format images, line by line.
- A parameter "header" was added to *mkobjects, mknoise, mk1dspec*, and *mk2dspec* so that a header data file could be added to the output image. Other header parameters are also added to the image header such as gain, rdnoise, and exptime.
- A "comments" parameter was added to various tasks to include/exclude comments in the header of the output image that describe the data parameters.

3.2.2. ASTUTIL package modifications

- A new task *gratings* has been added to the package. This task computes grating parameters; five of the seven parameters must be specified at one time.
- A new task *setjd* has been added to the package. *setjd* computes the geocentric, heliocentric, and integer local Julian dates from information given in the headers of the input list of images. This information is then listed or added to the image headers.
- A few changes were made to *setairmass*. The default value for "update" was changed to "yes"; an update field was added to the "show" messages; a warning is printed if both "show" and "update" are set to "no" indicating that nothing was done.

3.2.3. DIGIPHOT package modifications

A new version of the DIGIPHOT package was installed. Some changes were made to the existing APPHOT package used for aperture photometry, and those are mentioned below. But three new packages have also been installed, DAOPHOT, PHOTCAL, and PTOOLS.

DAOPHOT has been available for the past two years as an add-on package known as TESTPHOT. It is now part of the distributed system. The IRAF version of DAOPHOT, used to do photometry in crowded fields, has been a collaborative effort with Peter Stetson and Dennis Crabtree of the DAO. For the convenience of the many TESTPHOT users, changes to the package since the last release of TESTPHOT are summarized below.

PHOTCAL is the photometric calibration package for computing the transformations of the instrumental magnitudes output from APPHOT and DAOPHOT to the standard photometric system. This package has been a collaborative effort with Pedro Gigoux at CTIO.

PTOOLS is a package containing an assortment of tools for manipulating the data files produced from APPHOT and DAOPHOT. If users are using STSDAS TABLES format data files then they must first install the TABLES layered package. This package can be obtained from either STScI or NOAO (see `iraf/contrib` in the IRAF network archive).

3.2.4. DIGIPHOT.APPHOT package modifications

- The *apselect* task was replaced with the functionally equivalent *txdump* task. *txdump* allows the user to select fields of data from the output data files produced from APPHOT tasks and either simply list the extracted data or save it in another file.
- A new task called *pexamine* has been added to the package. *pexamine* is a general purpose tool for interactively examining and editing the data files produced from tasks in either APPHOT or DAOPHOT. This task is intended to complement the batch oriented task *txdump*.
- All of the APPHOT tasks will now query to verify the "datamin" and "datamax" values, if these values are used by the tasks.
- The time of the observation, i.e. UT, can now be carried over into the output data files, if a keyword in the image header contains this information.
- If there is bad data within the photometry aperture a value of INDEF will be stored in the data file for that magnitude.

3.2.5. DIGIPHOT.DAOPHOT (TESTPHOT) package modifications

This is a new package but for the convenience of the many users of the TESTPHOT add-on package, the changes to TESTPHOT between its last release and its installation into the DIGIPHOT package as DAOPHOT are summarized below.

- The *append*, *convert*, *dump*, *renumber*, *select*, and *sort* tasks were renamed to *pappend*, *pconvert*, *pdump*, *prenumber*, *pselect*, and *psort*.

- The "text" parameter was moved from *daopars* to the DAOPHOT package parameters.
- All the DAOPHOT routines were modified so that "psfrad", "fitrad", and "matchrad" are defined in terms of scale.
- The tasks *allstar*, *group*, *nstar*, *peak*, *psf*, and *substar* were all modified to include "datamin" and "datamax" in their verify routines.
- Support was added for a time of observation parameter to all the appropriate DAOPHOT tasks. If present, this time will be carried over into the output data files.
- All the DAOPHOT tasks except *psf* have been modified to accept lists of input and output files.
- The new *pexamine* task was added to this package. *pexamine* is a general purpose tool for interactively examining and editing the data files produced from tasks in either APPHOT or DAOPHOT. This task is intended to complement the batch oriented task *txdump*.
- Several changes were made to *psf*: the default PSF image header will now hold up to 66 stars (but it is still dependent on the environment variable `min_lenuserarea`); a check was added so that the same star can not be added to the PSF twice; potential PSF stars are now rejected if their sky or magnitude values are INDEF; a check was added so that stars with INDEF valued positions are treated as stars that were not found.
- *group* was modified so that the groups are output in y order instead of in order of the size of the group.
- Both *group* and *peak* were modified so that stars with INDEF centers are not written to the output file.

3.2.6. IMRED package modifications

The spectroscopic packages within the IMRED package have undergone quite a bit of work and reorganization. The spectroscopic packages are now ARGUS, CTIOSLIT, ECHELLE, HYDRA, IIDS, IRS, KPNOCOUDE, KPNOSLIT, and SPECRED. These packages are a collection of tasks from APEXTRACT and ONEDSPEC that are appropriate for the specific applications. All these packages use the new versions of the APEXTRACT and ONEDSPEC packages; the changes for APEXTRACT and ONEDSPEC are summarized below. Earlier versions of all these packages were released as the NEWIMRED add-on package. The NEWIMRED package is now defunct and the distributed system contains the latest versions of these packages.

The spectroscopic packages now contain "DO" tasks that are scripts that streamline the reduction process for the user. The "DO" tasks have been modified for each particular application.

The ARGUS package is for the reduction of data taken with the CTIO *argus* instrument. Its corresponding script task is *doargus*.

The CTIOSLIT package is similar to the ARGUS package except that is a collection of spectroscopic tasks used for general CTIO slit reductions. The *doslit* task allows for streamlined reductions.

The ECHELLE package has the addition of the *doecslit* task for simplied echelle reductions. The *dofoe* task has been added for the reduction of Fiber Optic Echelle (FOE) spectra.

The HYDRA package is used for the reduction of multifiber data taken at KPNO. The *dohydra* task has been customized for streamlining *nessie* and *hydra* reductions.

The KPNOCOUDE package has been customized for the KPNO Coude. The *doslit* task allows the user to go through the reduction process with as few keystrokes as possible. The *do3fiber* task is specialized for the 3 fiber (two arc and one object) instrument.

The KPNOSLIT package can be used for general slit reductions at KPNO. The *doslit* task in this package is similar to that in the other packages.

The SPECRED package is the old MSRED package, used for general multispectral reductions. This is a generic package that can be used for slit and multifiber/ aperture data. General *doslit* and *dofibers* tasks are available.

Several of the spectroscopic packages has a special task called *msresp1d* for creating 1d aperture response corrections from flat and throughput data. This task is specialized for multiaperture or multifiber data.

All of the "DO" tasks have reference manuals that are available in the network archive in the `iraf/docs` directory.

3.2.7. IMRED.CCDRED package modifications

- A new task *ccdinstrument* was added to the package. The purpose of this task is to help users create new CCD instrument translation files to use with the package.
- The *combine* task (as well as *darkcombine*, *flatcombine*, and *zerocombine*) is the same task as the new *imcombine* task in IMAGES. A few parameters were added for compatibility with the CCDRED tasks.
- A new parameter was added to *ccdproc* called "minreplace". Pixel values in flat fields below the value for "minreplace" are replaced by the same value (after overscan, zero, and dark corrections). This avoids dividing by zero problems.
- The default computation and output "pixeltype" in the package parameter for CCDRED are both real. Note that both can now be specified separately.
- The default parameters for *darkcombine* and *flatcombine* have been changed.

3.2.8. NOBSOLETE package added

This new package has been defined but currently no tasks reside in it. This package will be used as a repository for tasks that have been replaced by newer tasks in the NOAO packages. The NOBSOLETE tasks will then be removed at the time of the next release.

3.2.9. NPROTO package modifications

The old PROTO package was divided into two separate packages, one called PROTO that now resides in the IRAF core system and the other called NPROTO that resides in the NOAO package. The current NPROTO tasks are *binpairs*, *findgain*, *findthresh*, *iralign*, *irmatch1d*, *irmatch2d*, *irmosaic*, *linpol*, and *slitpic*. The *imedit*, *imexamine*, and *tvmark* tasks were moved to the IMAGES.TV package. The *ndprep* task was moved to ONEDSPEC. All other tasks were moved to the PROTO package at the core level.

- Two new tasks called *findgain* and *findthresh* were added to this package. *findgain* determines the gain and read noise of a CCD from a pair of dome flats and from a pair of bias/zero frames using the Janesick method. *findthresh* estimates the background noise level of the CCD using a sky frame and the gain and readnoise of the CCD.
- A new task called *linpol* has been added to the PROTO package. This task calculates the pixel-by-pixel fractional linear polarization and polarization angle for three or four images. The polarizer must be set at even multiples of a 45 degree position angle.
- The task *ndprep* used for CTIO 2DFRUTTI reductions was moved to the ONEDSPEC package.
- The task *toonedspec* was removed since its function can now be performed by the *onedspec.scopy* task.

3.2.10. ONEDSPEC package reductions

There have been significant changes to the ONEDSPEC package. A more detailed revisions summary is available in the IRAF network archive as file `onedv210.ps.Z` in the subdirectory `iraf/docs`.

The new package supports a variety of spectral image formats. The older formats can still be read. In particular the one dimensional "onedspec" and the two dimensional "multispec" format will still be acceptable as input. Note that the image naming syntax for the "onedspec" format using record number extensions is a separate issue and is still provided but only in the IMRED.IIDS and IMRED.IRS packages. Any new spectra created are either a one dimensional format using relatively simple keywords and a two or three dimensional format which treats each line of the image as a separate spectrum and uses a more complex world coordinate system and keywords. For the sake of discussion the two formats are still called "onedspec" and "multispec" though they are not equivalent to the earlier formats.

In addition, the one dimensional spectral tasks may also operate on two dimensional images. This is done by using the "dispaxis" keyword in the image header or a package dispaxis parameter if the keyword is missing to define the dispersion axis. In addition there is a summing parameter in the package to allow summing a number of lines or columns. If the spectra are wavelength calibrated long slit spectra, the product of the *longslit.transform* task, the wavelength information will also be properly handled. Thus, one may use *splot* or *specplot* for plotting such data without having to extract them to another format. If one wants to extract one dimensional spectra by summing columns or lines, as opposed to using the more complex APEXTRACT package, then one can simply use *scopy* (this effectively replaces *proto.toonedspec*)

Another major change to the package is that spectra no longer need to be interpolated to a uniform sampling in wavelength. The dispersion functions determined by *identify*/*reidentify* can now be carried along with the spectra in the image headers and recognized throughout the package and the IRAF core system. Thus the WCS has been fully implemented in the ONEDSPEC tasks and although much is to be gained by this it can cause problems with earlier IRAF systems as well as making it difficult to import data into non-IRAF software. But it is now possible to use *imcopy* with an image section on a spectrum, for instance, and have the wavelength information retained correctly.

A sinc interpolator is now available for those tasks doing geometric corrections or interpolations. This option can be selected by setting the "interp" parameter in the ONEDSPEC package parameters.

Other changes are summarized:

- The new task *deredden* corrects input spectra for interstellar extinction, or reddening.
- The new task *dopcor* corrects input spectra by removing a specified doppler velocity. The opposite sign can be used to add a doppler shift to a spectrum.
- The new task *fitprofs* fits 1D gaussian profiles to spectral lines or features in an image line or column. This is done noninteractively and driven by an input list of feature positions.
- The task *ndprep* was moved to this package from the PROTO package. *ndprep* is used for CTIO *2DFRUTTI* reductions.
- The new task *sapertures* adds or modifies beam numbers and aperture titles for selected apertures based on an aperture identification file.
- The new task *sarith* does spectral arithmetic and includes unary operators. The arithmetic can be performed on separate input spectra or on apertures within a multispec format image.
- The task *scombine* has been added to the package. This new task is similar to the new *imcombine* task in the IMAGES package but is specialized for spectral data.
- The new task *scopy* copies subsets of apertures and does format conversions between the different spectrum formats.
- The new task *sfit* fits spectra and outputs the fits in various ways. This includes a new feature to replace deviant points by the fit. Apertures in multispec and echelle format are fit independently.

- The tasks *addsets*, *batchred*, *bswitch*, *coincor*, *flatdiv*, *flatfit*, *subsets* and *sums* were moved to the IIDS and IRS packages.
- The task *combine* was removed with the all new *scombine* suggested in its place.
- The tasks *rebin*, *sflip* and *shedit* were removed from the package. Rebinning of spectra can now be done with *scopy* or *dispcor*. *scopy* and *imcopy* can now be used in place of *sflip*. And *hedit* is an alternative for *shedit*.
- *bplot* and *slist* have been modified to support the multispec format.
- The task *continuum* now does independent fits for multispec and echelle format spectra.
- There is now only one *dispcor* task doing the work of the three previous tasks *dispcor*, *msdispcor* and *ecdispcor*. Several of the parameters for this task have been changed. The old "recformat" and "records" parameters for supporting the old onedspec format have been removed except in the IIDS/IRS packages. A "linearize" parameter has been added for setting the interpolation to yes or no on output. The "interpolation" parameter was removed since this information is now read from the package parameter "interp". The "ignoreaps" parameter has been changed to "samedisp".
- *identify* and *reidentify* now treat multispec format spectra and two dimensional images as a unit allowing easy movement between different image lines or columns. The database is only updated upon exiting the image. The "j", "k", and "o" keys have been added to the interactive sessions to allow for scrolling through the different lines in a two dimensional image. The database format now uses aperture numbers rather than image sections for multispec format spectra.
- The task *specplot* has new parameters "apertures" and "bands" to select spectra for plotting in the multispec format.
- *splot* now allows deblending of any number of components and allows simultaneous fitting of a linear background. Several cursor keys have been redefined and colon commands have been added to fully support the new multispec format and to add even more flexibility to the task than before! Please see the help pages for details.
- The *reidentify* task is essentially a new task. The important changes are an interactive review option, the ability to add features from a line list, using the same reference spectrum for all other spectra in a 2D reference image rather than "tracing" (using the results of the last reidentification as the initial guess for the next one) across the image, and matching image lines/columns/apertures from a 2D reference image to other 2D images rather than "tracing" again.

3.2.11. RV package added

This is a new package installed with IRAF version 2.10. A version of this package, RV0, has been available as an add-on for the past year or so. The package contains one main task *fxcor* that performs a Fourier cross-correlation on the input list of spectra. The package supports the multispec format.

3.2.12. TWODSPEC package modifications

The old MULTISPEC package that has resided in the TWODSPEC package for years has been disabled. The code is still there for browsing, however, and the package can be resurrected easily if needed.

The *observatory* task was moved to the NOAO package itself. The *setairmass* task was moved to LONGSLIT. And the *setdisp* task is no longer needed; task or package parameters are used in its place.

Changes to the APEXTRACT and LONGSLIT packages are summarized below.

3.2.13. TWODSPEC.APEXTRACT package modifications

A new version, version 3, of the IRAF APEXTRACT package has been completed. A detailed revisions summary is available in the IRAF network archive (file `apexv210.ps.Z` in `iraf/docs`).

There were three goals for the new package: new and improved cleaning and variance weighting (optimal extraction) algorithms, the addition of recommended or desirable new tasks and algorithms (particularly to support large numbers of spectra from fiber and aperture mask instruments), and special support for the new image reduction scripts in the various IMRED packages.

The multispec format, the default image format for all spectroscopic packages except IIDS and IRS, is either 2D or 3D (the 3D format is new with this release). The 3D format is produced when the parameter "extras" is set to yes in the extraction tasks. The basic 2D format, which applies to the first plane of the 3D format, has each 1D aperture spectra extracted in a line. Thus, the first coordinate is the pixel coordinate along the spectra. The second coordinate refers to the separate apertures. The third coordinate contains extra information associated with the spectrum in the first plane. The extra planes are:

1: Primary spectra which are variance and/or cosmic ray cleaned
2: Spectra which are not weighted or cleaned
3: Sky spectra when using background subtraction
4: Estimated sigma of the extracted spectra

If background subtraction is not done then 4 moves down to plane 3. Thus:

```
output.ms[*,*,1] = all primary spectra
output.ms[*,5,1] = 5th aperture spectrum which is cleaned
output.ms[*,5,2] = raw/uncleaned 5th aperture spectrum
output.ms[*,5,3] = sky for 5th spectrum
output.ms[*,5,4] = sigma of 5th spectrum
```

The following summarizes the major new features and changes.

- New techniques for cleaning and variance weighting extracted spectra have been implemented.
- Special features for automatically numbering and identifying large numbers of apertures have been added.
- There is no longer a limit to the number of apertures that can be extracted.
- The new task *apall* integrates all the parameters used for one dimensional extraction of spectra.
- The new task *apfit* provides various types of fitting for two dimensional multiobject spectra.
- The new task *apflatten* creates flat fields from fiber and slitlet spectra.
- The new task *apmask* creates mask images from aperture definitions using the new pixel list image format. No tasks have yet been written, however, to use this new mask image format.
- The new tasks and algorithms, *aprecenter* and *apresize*, will automatically recenter and resize aperture definitions.
- The task *apio* is defunct. Its functionality has been replaced by the APEXTRACT package parameters.
- The task *apstrip* has been removed.

- Minor changes have been made to the old "AP" tasks to accommodate these new changes in the package; in some cases new parameters have been added to the task and in other cases new cursor options have been implemented. See the help pages for details.

3.2.14. TWODSPEC.LONGSLIT package modifications

- The "dispaxis" parameter was added to the package parameters. This value is used unless DISPAXIS is in the image header.

3.2.15. Glossary of new tasks in the NOAO packages

Task		Package		Description
addstar	-	daophot	-	Add artificial stars to an image using the computed psf
allstar	-	daophot	-	Group and fit psf to multiple stars simultaneously
apall	-	apextract	-	Extract 1D spectra (all parameters in one task)
apfit	-	apextract	-	Fit 2D spectra and output the fit, difference, or ratio
apflatten	-	apextract	-	Remove overall spectral and profile shapes from flat fields
apmask	-	apextract	-	Create and IRAF pixel list mask of the apertures
aprecenter	-	apextract	-	Recenter apertures
apresize	-	apextract	-	Resize apertures
ccdinstrument	-	ccdred	-	Review and edit instrument translation files
centerpars	-	daophot	-	Edit the centering algorithm parameters
chkconfig	-	photcal	-	Check the configuration file for grammar and syntax errors
continpars	-	rv	-	Edit continuum subtraction parameters
daofind	-	daophot	-	Find stars in an image using the DAO algorithm
daopars	-	daophot	-	Edit the daophot algorithms parameter set
datapars	-	daophot	-	Edit the data dependent parameters
deredden	-	onedspec	-	Apply interstellar extinction correction
do3fiber	-	kpnocoude	-	Process KPNO coude three fiber spectra
doargus	-	argus	-	Process ARGUS spectra
doecslit	-	echelle	-	Process Echelle slit spectra
dofibers	-	specred	-	Process fiber spectra
dofoe	-	echelle	-	Process Fiber Optic Echelle (FOE) spectra
dohydra	-	hydra	-	Process HYDRA spectra
dopcor	-	onedspec	-	Apply doppler corrections
doslit	-	ctioslit	-	Process CTIO slit spectra
doslit	-	kpnocoude	-	Process KPNO coude slit spectra
doslit	-	kpnoslit	-	Process slit spectra
doslit	-	specred	-	Process slit spectra
evalfit	-	photcal	-	Compute the standard indices by evaluating the fit
filtpars	-	rv	-	Edit the filter function parameters
findgain	-	nproto	-	Estimate the gain and readnoise of a CCD
findthresh	-	nproto	-	Estimate a CCD’s sky noise from the gain and readnoise
fitparams	-	photcal	-	Compute the parameters of the transformation equations
fitprofs	-	onedspec	-	Fit gaussian profiles
fitskypars	-	daophot	-	Edit the sky fitting algorithm parameters
fxcor	-	rv	-	Radial velocities via Fourier cross correlation
gratings	-	astutil	-	Compute and print grating parameters
group	-	daophot	-	Group stars based on positional overlap and signal/noise
grpselect	-	daophot	-	Select groups of a specified size from a daophot database
invertfit	-	photcal	-	Compute the standard indices by inverting the fit
istable	-	ptools	-	Is a file a table or text database file ?
linpol	-	nproto	-	Calculate polarization frames and Stoke’s parameters

keywpars	-	rv	-	Translate the image header keywords used in RV package
mkcatalog	-	photcal	-	Type in a standard star catalog or observations file
mkconfig	-	photcal	-	Prepare a configuration file
mkechelle	-	artdata	-	Make artificial 1D and 2D echelle spectra
mkexamples	-	artdata	-	Make artificial data examples
mkheader	-	artdata	-	Append/replace header parameters
mkimsets	-	photcal	-	Prepare an image set file for input to (mk)(n)obsfile
mk(n)obsfile	-	photcal	-	Prepare a single (multi)-starfield observations file from apphot/daophot output
mkphotcors	-	photcal	-	Type in/check any photometric corrections files required by mk(n)obsfile
msresp1d	-	argus	-	Create 1D response spectra from flat field and sky spectra
msresp1d	-	hydra	-	Create 1D response spectra from flat field and sky spectra
msresp1d	-	kpnocoude	-	Create fiber response spectra from flat field and sky spectra
msresp1d	-	specred	-	Create 1D response spectra from flat field and sky spectra
nstar	-	daophot	-	Fit the psf to groups of stars simultaneously
obsfile	-	photcal	-	Prepare a single (multi)-starfield observations file from a user-created text file
pappend	-	daophot	-	Concatenate a list of daophot databases
pappend	-	ptools	-	Concatenate a list of apphot/daophot databases
pconvert	-	daophot	-	Convert a text database to a tables database
pconvert	-	ptools	-	Convert from an apphot/daophot text to tables database
pdump	-	daophot	-	Print selected fields from a list of daophot databases
pdump	-	ptools	-	Print selected columns of a list of daophot/apphot databases
peak	-	daophot	-	Fit the psf to single stars
pexamine	-	apphot	-	Interactively examine or edit an apphot output file
pexamine	-	daophot	-	Interactively examine and edit a daophot database
pexamine	-	ptools	-	Interactively examine and edit an apphot/daophot database
phot	-	daophot	-	Compute sky values and initial magnitudes for a list of stars
photpars	-	daophot	-	Edit the photometry parameters
prenumber	-	daophot	-	Renumber stars in a daophot database
prenumber	-	ptools	-	Renumber a list of apphot/daophot databases
pselect	-	daophot	-	Select records from a daophot database
pselect	-	ptools	-	Select records from a list of apphot/daophot databases
psf	-	daophot	-	Fit the point spread function
psort	-	daophot	-	Sort a daophot database
psort	-	ptools	-	Sort a list of apphot/daophot databases
sapertures	-	onedspec	-	Set or change aperture header information
sarith	-	onedspec	-	Spectrum arithmetic
scombine	-	onedspec	-	Combine spectra having different wavelength ranges
scopy	-	onedspec	-	Select and copy apertures in different spectral formats
seepsf	-	daophot	-	Compute an image of the point spread function
setjd	-	astutil	-	Compute and set Julian dates in images
sfit	-	onedspec	-	Fit spectra and output fit, ratio, or difference
substar	-	daophot	-	Subtract the fitted stars from the original image
tbappend	-	ptools	-	Concatenate a list of apphot/daophot tables databases
tbdump	-	ptools	-	Print selected columns of a list of tables databases
tbrenumber	-	ptools	-	Renumber a list of apphot/daophot tables databases
tbselect	-	ptools	-	Select records from a list of apphot/daophot tables databases
tbsort	-	ptools	-	Sort a list of apphot/daophot tables databases
txappend	-	ptools	-	Concatenate a list of apphot/daophot text databases
txdump	-	apphot	-	Dump selected fields from an apphot output file
txdump	-	ptools	-	Print selected columns of a list of apphot/daophot text databases
txrenumber	-	ptools	-	Renumber a list of apphot/daophot text databases

txselect	-	ptools	-	Select records from a list of apphot/daophot text databases
txsort	-	ptools	-	Sort a list of apphot/daophot text databases

4. Programming Environment Revisions

4.1. Compatibility Issues

V2.10 IRAF requires that any local IRAF programs external to the V2.10 system (such as layered packages or locally written tasks) be *fully recompiled* if they are relinked against V2.10. The problem arises only if the programs are relinked; external programs will continue to run after V2.10 is installed, but linker errors will be seen if the programs are relinked without being fully recompiled. This is because the internal names of some important system routines were changed in V2.10 to avoid name clashes with host system routines. For example, the SPP procedure "rename" is now translated to "xfrnam" when the SPP code it appears in is compiled.

As always, actual interface changes affecting existing source code were very few. The macro "E" in `<math.h>` was renamed to "BASE_E" to minimize the chance of an accidental name collision. The calling sequence for the *onentry* procedure (ETC) was changed, but since this is a little used system procedure very few tasks should be affected. A number of new procedures were added to MTIO and the syntax of a magtape device has changed; old applications should be modified to use the MTIO procedures to operate upon magtape device names.

These and other revisions affecting the programming environment are discussed in more detail below.

4.2. Portability Issues

The V2.10 UNIX/IRAF kernel now includes "#ifdef SYSV" support for System V UNIX, making it easier to port IRAF to SysV based systems. The UNIX/IRAF HSI (host system interface) is still not as portable to UNIX variants as it could be, but at this point it is easier for us to make the minor revisions required for a new port than to further refine the HSI. The disadvantage is that it is harder than it should be for people in the community to do their own IRAF ports, due to the level of IRAF expertise required to tune the code. Someday we plan to generate a generic-UNIX HSI. Note that these comments pertain only to the few thousand lines of code in the HSI - the bulk of the IRAF code is 100% portable (identical on all IRAF systems) as it has always been.

The recent port of IRAF to A/UX (the Apple Macintosh UNIX) is interesting from a portability standpoint because we used the publically available Fortran to C translator *f2c* plus the GNU C compiler *gcc* to compile all the SPP and Fortran code in IRAF. This was remarkably successful and means that the IRAF code is now portable to any system with a C compiler. In the process of performing these compilations a few dozen minor bugs were found by the static compile time checking performed by *f2c* and *gcc*. The IRAF C code was run through *gcc* with ANSI mode enabled, and hence should now be ANSI-C compatible. The GNU debugger *gdb* proved to be an effective tool for debugging of IRAF code compiled with *gcc*.

4.3. Software Tools

4.3.1. LOGIPC feature

A new facility has been added to UNIX/IRAF (all systems) for debugging interprocess communication (IPC). This feature will also be useful for debugging tasks standalone, particularly in cases where a bug seen when running a task from the CL is difficult to reproduce when the task is run standalone. The new facility allows one to carry out a normal IRAF session while transparently logging all interprocess communications. Each process can then be rerun individually using the logged IPC to exactly duplicate the functioning of the process to, e.g., examine the program operation in detail under a debugger.

The facility is this: if LOGIPC is defined in the host environment when an iraf process is started, the process will create two files *pid*`.in` and *pid*`.out`, where *pid* is the process id. Everything read from the IPC input file is copied to the `.in` file, and everything written to the IPC output (i.e., sent back to the CL) is copied to the `.out` file. This is done only for connected subprocesses. It will work for any connected subprocess, e.g., normal cached processes and graphics subkernels in both foreground and background CLs, but not the i/o of the CL itself since the CL is not driven by IPC.

The IPC streams saved are an exact binary copy of whatever was sent via IPC, including the binary IPC packet headers, and binary graphics data, and so on. All the IPC communications used to start up the process, run zero or more tasks, and close the process down will be logged. Most IPC traffic is textual in nature so it will usually be possible to examine the IPC files with a file pager, although the results may be less than satisfactory if binary data such as a graphics metacode stream is logged. It is not necessary to examine the IPC files to use them for process execution or debugging.

A particularly interesting use of this feature is to allow a process to be run under the CL in the normal fashion, then rerun under a host level debugger using the saved IPC input to duplicate the input and actions of the process when run under the CL. For example,

```
% setenv LOGIPC
% cl
cl> dir
cl> logout
% unsetenv LOGIPC
```

will run the CL, saving the IPC of all subprocesses, e.g. `x_system.e`. We can then run the system process manually, using the saved IPC input:

```
% $iraf/bin/x_system.e -c < pid.in
```

To run the process under *adb* or *dbx*, using the saved input:

```
% adb $iraf/bin/x_system.e
:r <pid.in -c
```

or

```
% dbx $iraf/bin/x_system.e
dbx> r -c < pid.in
```

Note that the redirection has to be given first for *adb*, and last for *dbx*. This also works for *gdb* using the *run* command. Some implementations of *adb* are picky about syntax and will not allow a space between the "<" and the process id in the :r command.

Running a task in this way is not identical to running the task standalone, e.g. using a *dparam* file for parameter input, because the full runtime context of the process as run under the CL is reproduced by LOGIPC but not when the task is run standalone. The differences are subtle but can be important when trying to reproduce a bug seen when the process is run under the CL. It is often the case that a bug seen when a task is run from the CL will disappear when the task is run standalone, but in most cases LOGIPC will duplicate the bug.

4.3.2. XC changes

The *xc* program (IRAF compiler-linker) underwent the following changes for V2.10, in addition to the usual host-system specific changes required to support new host compiler versions.

Multiple "-p pkgname" arguments are now supported. These are used when compiling a module which uses multiple package environments, e.g.,

```
cl> xc -p noao -p tables foo.x
```

Each package environment may define package environment variables, directories to be searched for include files and libraries, and so on. Package environments are used by the IRAF layered packages.

Host libraries named on the command line using the -l switch (e.g., "-lresolv") are now searched after the IRAF system libraries, rather than before as in previous versions of *xc*. If the host library is referenced by absolute pathname it is still searched before the IRAF libraries, since the command line order determines the search order.

4.3.3. SPPLINT code checker

A static code checker utility *spplint* has been developed for checking SPP programs. This uses the SPP translation utilities in IRAF to convert SPP to Fortran, *f2c* to generate C code, and *lint* to check the C code. The code is checked in various ways at all phases of translation. Some of the most useful checking are performed by *f2c*, which checks the number and type of arguments in all calls to IRAF VOS library procedures. In other words, *spplint* will determine if an IRAF library procedure is called incorrectly, as well as perform a variety of other checks.

The *spplint* utility is not included in the main IRAF release, but is available separately. Contact the IRAF project for information on the availability of this and other optional code development utilities.

4.3.4. Multiple architecture support

Multiple architecture support is a way of using multiple sets of compiled program binaries within a single copy of IRAF. Multiple sets of binaries are used to support different machine architectures (e.g. sparc and Motorola), different compiler options (floating point options, vectorization options, profiling options), different compilers, and so on.

The command for checking the architecture of the IRAF core system or a layered package has been changed from *showfloat* to *arch* to reflect the fact that multiple architecture support is no longer used merely to select the floating point option.

```
cl> mkpkg arch
sparc
```

The default architecture of the distributed system is "generic", meaning no specific architecture has been set (the source tree is generic, not configured for any particular architecture).

It is suggested that developers of layered software for IRAF adopt this same convention in their root *mkpkg* files.

4.3.5. Package environments

All the HSI software utilities now permit multiple "-p pkgname" package environment references. The host `PKGENV` environment variable now also permits multiple package environments to be referenced, e.g.

```
% setenv PKGENV "noao tables xray"
```

The package names should be whitespace delimited (`PKGENV` is used to avoid having to give the "-p" flags on the *mkpkg* or *xc* command line). To successfully load a package enironment, the package root directory must be defined in `hlib$extern.pkg` or in the user's host environment. A host environment definition will override the value given in extern.pkg.

4.3.6. Finding module sources

IRAF V2.10 includes a "tags" file in the IRAF root directory to aid software development. This file contains an index to all procedures in the IRAF VOS and HSI and can be used with host editors such as *vi* to rapidly find and display the source for IRAF system procedures. Note that the names of the kernel procedures are given in upper case, e.g., "ZOPNBF", whereas the names of the VOS procedures are given in lower case. To use the tags file with *vi*, start the editor at the IRAF root directory and while in the editor, type a command such as ":ta foo" to view the source for procedure *foo*.

4.4. Programming Interface Changes

4.4.1. IEEE floating support

Modifications were made to the IEEE floating conversion routines in the OSB package to support NaN mapping. This is a low level package used by, e.g., the MII package in ETC. The interface as it currently stands is summarized below.

```
     ieepak[rd] (datum)               # pack scalar value
     ieeupk[rd] (datum)               # unpack scalar value
    ieevpak[rd] (native, ieee, nelem) # pack vector
    ieevupk[rd] (ieee, native, nelem) # unpack vector
 iee[sg]nan[rd] (NaN)                 # set/get NaN value
     ieemap[rd] (mapin, mapout)       # enable/disable NaN mapping
    ieestat[rd] (nin, nout)           # get count of NaN values
   ieezstat[rd] ()                    # zero NaN counters
```

The new or modified routines are *ieesnan*, *ieegnan*, *ieemap*, *ieestat*, and *ieezstat*. By NaN (not-a-number) we refer collectively to all IEEE non-arithmetic values, not just IEEE NaN. The routines *ieesnan* and *ieegnan* set or get the native floating value used to replace NaNs or overflows occurring when converting IEEE to the native floating format (any floating value will do, e.g., zero or INDEF). If NaN mapping is enabled, the *ieestat* routines may be used to determine the number of input or output NaN conversions occurring since the last call to *ieezstat*. Both real and double versions of all routines are provided.

The NaN mapping enable switch and statistics counters are undefined at process startup; programs which use the IEEE conversion package should call *ieemap* to enable or disable NaN mapping, and *ieezstat* to initialize the statistics counters.

4.4.2. MATH libraries

The following changes were made to the MATH libraries in the IRAF core system. Refer to the online help pages of the affected routines for detailed information.

- The one-dimensional image interpolation library **iminterp** was modified to add support for sinc interpolation.
- A new library **nlfit** was added for non-linear function fitting. An interactive graphics front end to this was also added in XTOOLS.
- The name of the symbol `E` in `<math.h>` was changed to `BASE_E` to minimize the chance of name clashes.

4.4.3. CLIO interface

The CLIO (command language i/o) interface underwent the following changes for version 2.10 IRAF.

- A README file was added to the source directory containing an up to date interface summary.
- The routines *clgpset* and *clppset* (get/put string valued parameter) were renamed *clgpseta* and *clppseta*. The old procedures were retained for compatibility but are now obsolete and may at some point in the future disappear or be reused for some other function.
- Two new routines *cllpset* and *clepset* were added for listing and editing psets (parameter sets).

The calling sequences for the new pset routines are as follows.

```
cllpset (pp, fd, format)    # list pset
clepset (pp)                # edit pset
```

These new routines are still considered experimental and should be avoided or used with caution (they could change).

Internal to the CLIO code, the CLIO parameter caching package underwent minor changes to add a new *clc_compress* routine and improve pset handling, as part of the minilanguage support effort.

4.4.4. ETC interface

The ETC interface contains miscellaneous system interface routines. The ETC interface underwent the following changes for V2.10 IRAF.

4.4.4.1. Calling sequence for onentry changed

The calling sequence for the *onentry* routine was changed. The new calling sequence is as follows.

```
action = onentry (prtype, bkgfile, cmd)
```

The *onentry* procedure is an optional procedure which is called when an IRAF process starts up. Normally the onentry procedure is a no-op, passing control to the IRAF main in-task interpreter. Custom IRAF applications, e.g., the CL, have a special *onentry* procedure which replaces the default in-task interpreter.

The change made to the *onentry* calling sequence was the addition of the additional argument *cmd*. This argument receives the command line used to invoke the IRAF process at the host level. The application can parse this command line to extract arguments, allowing the IRAF process to operate as a host program (it is already possible to call any IRAF task from the host level, but use of an *onentry* procedure allows the in-task interpreter to be bypassed and gives the application control over parsing the command line).

See `etc$onentry.x` for additional information on how to use this procedure.

4.4.4.2. New onerror and onexit procedures

Two new procedures *onerror_remove* and *onexit_remove* were added. These have the following calling sequences.

```
onerror_remove (proc)      # remove posted onerror procedure
 onexit_remove (proc)      # remove posted onexit procedure
```

The new routines are used to remove *onerror* or *onexit* procedures posted by a task during task execution. Such user procedures are called if the task aborts (*onerror* procedures) or during normal task exit (*onexit* procedures). Formerly there was no way to "unpost" the procedures other than by the normal cleanup occurring during task termination.

4.4.4.3. New gqsort routine

A new quick-sort routine *gqsort* (generalized quick sort) was added. This has the following calling sequence.

```
gqsort (x, nelem, compare, client_data)
```

gqsort is identical to the old *qsort* routine except for the addition of the fourth argument *client_data*. This is an integer value which is passed on to the *compare* procedure during sorting to compare two data values. The *compare* routine is called as follows.

```
result = compare (client_data, x1, x2)
```

The new routine eliminates the need to use a common area to pass information to the compare routine, as was often necessary with *qsort*.

4.4.5. FIO interface

The FIO interface (file i/o) underwent minor changes to fix some bugs affecting pushed back data. The `F_UNREAD` file status parameter will now correctly report pushed back data as well as any buffered input file data. The `F_CANCEL` file set option will now cancel any pushed back data.

4.4.5.1. Nonblocking terminal i/o

A new nonblocking form of raw mode terminal input has been added. This permits polling the terminal for input without blocking (suspending process execution) if no input is available. In a character read, EOF is returned if no input is available otherwise the character value is returned. An example illustrating the use of nonblocking terminal i/o follows.

```
include <fset.h>

task foo

procedure foo()

int     fd, ch
int     getci()

begin
        fd = STDIN
        call fseti (fd, F_IOMODE, IO_RAW+IO_NDELAY)

        repeat {
            if (getci(fd,ch) == EOF)
                call printf ("no pending input\r\n")
            else {
                call printf ("ch = %03o\r\n")
                    call pargi (ch)
            }
            call sleep (1)
        } until (ch == '\004' || ch == 'q')

        call fseti (fd, F_IOMODE, IO_NORMAL)
end
```

This sample program sets the terminal i/o mode to nonblocking raw mode and polls the terminal once per second, printing the character value in octal if a character is typed on the terminal, exiting when ctrl/d or 'q' is typed. Note that in raw mode characters such as ctrl/d or ctrl/c are passed through as data and do not get mapped to EOF, generate an interrupt, and so on. Raw mode i/o such as this will work both when running a task under the CL and standalone, and in combination with IRAF networking (e.g. to access a remote device).

Nonblocking terminal input is used in applications which run continuously but which we wish to be able to control interactively.

4.4.6. FMTIO interface

The FMTIO interface (formatted i/o) is used to format output text or decode input text. The V2.10 release adds two new *printf* output formats, `%H` and `%M`. These are used to print numbers in hours-minutes-seconds or minutes-seconds format and are equivalent to the older output formats `%h` and `%m` except that the number is first divided by 15. This converts degrees to hours allowing values given in units of degrees to be printed as hours with just a change of the output format. In other words, given a number N in units of degrees, `%H` would print the number in hours-minutes-seconds, i.e., "hh:mm:ss.ss", whereas `%h` would print the same number as degrees-minutes-seconds, "dd:mm:ss.ss". The `%m` formats are similar except that only two of the three fields are printed.

4.4.7. GTY interface

The GTY interface is a generalized version of that portion of the older TTY interface dealing with termcap format files. The TTY code which accesses termcap format files has been extracted to form the new GTY interface, allowing arbitrary termcap format files to be accessed by filename, unlike TTY which returns TTY descriptors given the termcap or graphcap device name. GTY was contributed by Skip Schaller of Steward Observatory.

```
           gty = gtyopen (termcap_file, device, ufields)
                 gtyclose (gty)
             cp = gtycaps (gty)
     pval = gtyget[bir] (gty, cap)
       nchars = gtygets (gty, cap, outstr, maxch)
```

The *gtyopen* call returns the GTY descriptor for the named *device* from the file *termcap_file*. The *ufields* string may be either NULL or a list of colon-delimited device capabilities, which will override the corresponding capabilities in the device entry given in the termcap file. If *termcap_file* is the null string *ufields* is taken to be the device entry for the named device. The *gtycaps* routine may be used to get the entire device entry as a string, whereas the *gtyget* and *gtygets* routines are used to get the values of individual capabilities or parameters.

4.4.8. MTIO interface

MTIO is the magtape i/o interface. The magtape i/o subsystem was extensively revised for V2.10 IRAF, as documented earlier in this revisions summary. The VOS level MTIO interface itself was not changed other than to add a few new routines. The *tapecap* facility is new in V2.10. The revisions to the host level magtape driver ZFIOMT required that the calling sequences of some of the interface routines be changed.

4.4.8.1. MTIO applications programming interface

The current MTIO applications programming interface is summarized below. Most of these routines are new: the old routines are *mtfile*, *mtopen*, *mtrewind* and *mtposition*.

```
        yes|no = mtfile (fname)
  yes|no = mtneedfileno (mtname)
            gty = mtcap (mtname)
                mtfname (mtname, file, outstr, maxch)

                mtparse (mtname, device, fileno, recno, attrl, maxch)
               mtencode (mtname, maxch, device, fileno, recno, attrl)

            fd = mtopen (mtname, acmode, bufsize)
               mtrewind (mtname, initcache)
             mtposition (mtname, file, record)
```

The *mtfile* routine is used to test whether the given filename is a magtape file or something else, i.e., a disk file. *mtneedfileno* tests whether a file number has been specified in *mtname* (e.g., to determine whether or not to query for a file number parameter). *mtfname* takes *mtname* and a file number and constructs a new magtape device name in the output string. *mtparse* parses a magtape device name into its component parts, and *mtencode* does the reverse. *mtcap* returns the GTY descriptor of the tapecap entry for the device. *gtyclose* should be used to free this descriptor when it is no longer needed.

Some older magtape applications programs parse the magtape device name directly, looking for characters like ‘[’. These old programs are making assumptions about the form of a magtape device name which are probably no longer true. Such old applications should be rewritten to use the new MTIO procedures for all magtape device name operations.

4.4.8.2. MTIO system procedures

The MTIO interface also includes a number of procedures intended for use in systems code. These are summarized in the table below.

```
  mtallocate (mtname)
mtdeallocate (mtname, rewind_tape)
    mtstatus (out, mtname)
     mtclean (level, stale, out)
```

The only new routine here is *mtclean*. This is called by the *mtclean* task in the V2.10 SYSTEM package and is used to scan the magtape status file storage area to delete old magtape position status files. Prior to V2.10 these files were stored in the user's UPARM directory, but in V2.10 the files are stored in `/tmp` so that multiple IRAF sessions will share the same tape position information. A special task is needed to delete old position files in order to protect against, e.g., one user deleting another user's device position file while the device is actively in use.

4.4.8.3. Magtape driver procedures

All access to the physical magtape device is via the host level IRAF magtape device driver ZFIOMT. The driver procedures had to be revised for V2.10 to add support for the tapecap facility and to accommodate changes in the way the device position is maintained. The new driver procedures are summarized below.

```
zzopmt (device, acmode, devcap, devpos, newfile, chan)
zzrdmt (chan, buf, maxbytes, offset)
zzwrmt (chan, buf, nbytes, offset)
zzwtmt (chan, devpos, status)
zzstmt (chan, param, value)
zzclmt (chan, devpos, status)
zzrwmt (device, devcap, status)
```

The corresponding KI (kernel interface) routines were also modified to reflect the new calling sequences. A result of this is that IRAF networking for magtape devices is incompatible between V2.10 and earlier versions of IRAF.

The host level device driver procedures are not normally called directly in applications code (applications use *mtopen*). Refer to the comments in the source file `os$zfiomt.c` for additional details.

4.4.9. MWCS interface

The MWCS interface provides world coordinate system support for the IRAF system and applications. The main changes in the MWCS interface for V2.10 were bug fixes and semantic changes, e.g. various restrictions having to do with WCS attributes were removed, new function drivers were added, and so on. These changes are documented elsewhere in this revisions summary.

The only interface change affecting MWCS was the addition of the new MWCS procedure *mw_show*.

```
mw_show (mw, fd, what)
```

This is used to dump a MWCS descriptor to the given output file, and is useful for examining MWCS descriptors while debugging applications.

5. Getting Copies of this Revisions Summary

Additional copies of this revisions summary are available via anonymous ftp from node `iraf.noao.edu` in the directory `iraf/v210`, or via email from `iraf@noao.edu`.