IRAF Newsletter

Number 14 *April 1998*

National Optical Astronomy Observatories, 950 N. Cherry, P.O. Box 26732, Tucson, AZ 85726

Table of Contents

IRAF Update

It has been quite a while since our last IRAF Newsletter. This was not because there has been little news, quite the contrary! Rather, with the phenomenal success of the Internet in the last several years, Internet-based facilities such as the IRAF Web pages and the *adass.iraf* mailing lists have proven to be a more timely and convenient way to provide up to date information about IRAF. See the article *The Online IRAF News* elsewhere in this Newsletter for a summary of these Internet-based facilities.

Although the Internet-based services have largely taken over the job of providing day to day news about new developments as well as online access to reference documentation, we feel that a periodically produced Newsletter is still a good format for summarizing new developments in an organized way at a longer timescale. It is particularly nice if one can print off a copy and read it efficiently away from the computer screen. Currently we are pursuing a combined strategy of trying to put out a Newsletter whenever we have a major release, while using the Internet to disseminate daily news of new developments in between major releases.

We are considering not mailing printed copies of the Newsletter and instead only providing PostScript, PDF, and browsable HTML versions on the Internet. Please let us know if you have an opinion, pro or con, on whether printed mailings are worthwhile. See *newslet_14.ps.gz* on *iraf.noao.edu* in the */iraf/docs/* directory for the PostScript version of this Newsletter. Additional printed copies are available upon request to anyone who asks for one. Drop us an email at *iraf@noao.edu* if you would like us to send you an extra copy or two.

Since our last Newsletter we have had several smaller releases of IRAF (not necessarily for all platforms),

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

some major new IRAF ports (e.g., Linux/IRAF, and the DEC Alpha and SGI versions of IRAF), plus several patch releases. Lots of new science applications software was released as layered packages during this period as well. X11IRAF was developed and released, including many enhancements to **Ximtool**. **SAOtng**, a variant of **Ximtool** and the successor to **SAOimage**, was introduced by SAO.

In the fall of 1997 we saw a major new release of IRAF, Version 2.11, followed by platform upgrades to make IRAF V2.11 available for all supported platforms (these upgrades are just winding up in late spring 1998). A corresponding major new release of the IRAF layered packages TABLES and STSDAS (V2.0), adding support for the new HST instruments STIS and NICMOS, was released by STScI.

Many of the features of the new IRAF V2.11 release are covered in other articles in this Newsletter. Release notes are available on the Web at *http://iraf.noao.edu/v211revs.html* or the file *iraf/v211/v211revs.txt* in the IRAF archives. Detailed notes on system changes are included in the distribution in the *iraf/local/notes.v211* file or in the *iraf/v211/sysnotes.v211* file in the IRAF archives. Updated installation materials are provided in the individual distribution directories (see */iraf/v211* in the IRAF archives) as these releases become available.

In the summer of 1997 the IRAF group at NOAO introduced the *NOAO Mosaic Data Handling System* (Mosaic DHS), a data acquisition, quick look, and data reduction system for Mosaic CCD instruments. The new data handling system and the NOAO Mosaic have been in use on Kitt Peak since the fall of 1997, and several other observatories have since adopted the Mosaic DHS for use with their own Mosaic instruments. Included is a new IRAF package, **mscred**, for reducing data from CCD Mosaic instruments.

This Mosaic DHS uses as its internal framework a modular and flexible new architecture based on a message bus and distributed objects. This was developed during 1997 as part of the NASA ADP funded *Open IRAF initiative*. As we proceed into 1998 the Open IRAF initative, and a GUI and image display effort funded by a NASA AISR grant, are becoming the central focus of the IRAF group in Tucson. We are excited by the potential of this new technology to change the way we do astronomy software, meanwhile giving us a path towards modernizing the IRAF system. These efforts will eventually result in a major reworking of the IRAF environment and give us what will essentially be a new system, while preserving the many man-years we have invested in existing high quality science applications.

Doug Tody

IRAF Project Awarded NASA Grants

In late 1996 the IRAF project was awarded a three year NASA ADP grant to support the Open IRAF initiative. Work on the grant began in 1997 and will run through 1999. This project represents a collaboration of the IRAF group at NOAO and the major IRAF development sites within NASA, including HST, AXAF, SAO, and CEA.

The Open IRAF initiative will evolve and enhance IRAF in many ways—to make IRAF more modular so that pieces of the system can be used as distinct products separate from the rest of IRAF, to allow better integration of IRAF with non-IRAF software and data formats, and to improve the support for user software development. These enhancements will include improved facilities for host execution of IRAF tasks and scripts, multilanguage support, support for host-callable IRAF libraries, and support for dynamically loadable modules. The multilanguage support will allow software to be developed in any of serveral popular programming languages including C, Fortran, and probably C++ or Java.

Open IRAF will aid our efforts to develop technology for distributed objects, messaging, and modular object components. This work is critical if IRAF is to continue to be viable in the future, and we are grateful to NASA for making it possible for this work to proceed. We appreciate receiving outside money to help NOAO support and develop IRAF, especially in these tight budget times.

The first phase of Open IRAF was completed in 1997. The primary goal of Phase 1 was to provide limited Open IRAF development to support certain major new IRAF-based projects. These projects had a fixed schedule and had to be completed and released to the public in mid-1997, which placed a constraint on the work we could do on Open IRAF in the first year. Substantial research was done on distributed objects and messaging. A prototype message bus facility was implemented and is in use now in the NOAO Mosaic Data Handling System. A paper summarizing this work was presented at ADASS 97.

Other FY97 Open IRAF work included supporting and extending the prototype/interim C bindings for the IRAF V2.11 release and the associated STSDAS V2.0 release (providing HST STIS/NICMOS calibrations). A host-callable image display interface library (the Client Display Library; see article elsewhere in this Newsletter) was implemented. This is part of the Open IRAF effort to enhance our facilities for calling IRAF components at the host level, e.g., IRAF tasks and libraries, or large components such as the **Ximtool** image visualization tool. The programming tools **Xc** and **Mkpkg** were extended to better support C language programming. Support was enhanced for standard data formats that can be shared by IRAF and non-IRAF software, most notably FITS.

The second phase of Open IRAF development is just now getting underway. This will concentrate on repacking the IRAF host system interface and build tools, host callable libraries, component packaging (a more flexible way of calling IRAF programs from different environments), and multi-language support, including support of the prototype language bindings in IRAF. In subsequent years we will continue the work on messaging and distributed objects, which will eventually lead (several years from now) to a major restructuring of the IRAF system.

IRAF, in partnership with the Smithsonian Astrophysical Observatory, has also been awarded a NASA AISR grant for the development of our image display and GUI technology. This grant from the NASA Applied Information Systems Research program (AISR) is for developing the "plug-in extensible image server". Our intention is that this new image display facility will eventually replace and supplant both **Ximtool** and **SAOtng**. The new display server will be based on the current IRAF Object Manager GUI technology and on the **Ximtool** and **SAOtng** display engines, but will be a major step beyond both. Enhancements to our GUI infrastructure should be an additional benefit of developing the new image server technology.

To quote from the grant proposal: "...we describe the next step in our plan to evolve our existing software into a fully extensible, cross-platform image display server that can run stand-alone or be integrated seamlessly into astronomical analysis systems. We will build a Plug-in Image Extension (PIE) server for astronomy, consisting of a modular image display engine that can be customized using "plug-in" technology. We will create plug-ins that reproduce all the current functionality of **SAOtng** and more. We also will devise a messaging system and a set of distributed, shared data objects to support integrating the PIE server into astronomical analysis systems. Finally, we will migrate our PIE server, plug-ins, and messaging software from the Unix and the X Window System to a platform-independent architecture that utilizes Tcl/Tk and Java."

The PIE image server will provide the engine for the Real-Time Display being developed for the NOAO CCD Mosaic project. In combination with the Open IRAF work, these grants will be a major step towards a future distributed object, component based IRAF system.

We are grateful to NASA for their continued support of the IRAF project.

Doug Tody

The Online IRAF News

These days the printed IRAF Newsletter is published infrequently as the Internet is a better medium for handy reference and for timely information about IRAF and IRAF projects (the printed newsletter is still useful however for documenting major software releases).

The IRAF Web pages (*http://iraf.noao.edu/*) provide a handy, up to date reference guide for IRAF. The IRAF home page is updated frequently with "headlines" noting recent events with a link to an article on each. The IRAF Web pages contain many other useful resources, such as the IRAF FAQ (frequently asked questions) list, and links for downloading software.

A still underutilized resource are the *adass.iraf* newsgroups. Most people have given up trying to read these newsgroups in their USENET form due to all the SPAM postings; although we do cancel SPAMs, it takes a few hours for the cancels to propagate through USENET space. However, each newsgroup is gatewayed to a *moderated* listserver-hosted mailing list. Subscribing to these mailing lists is a good way to obtain updates on various aspects of the IRAF software: major announcements, system management issues, science applications, bug notices, programming, and so on. The traffic on the moderated lists is light, which is probably just as well for a mailing list. The following are the major IRAF-related ADASS newsgroups:

adass.general	Items of general interest.
adass.misc	General discussion of astronomical software.
adass.conference	Software conference announcements.
adass.iraf.announce	Announcements of new IRAF software, documentation, etc.
adass.iraf.buglog	Project bug notices (this is not a discussion group)
adass.iraf.applications	Discussion of IRAF applictions software.
adass.iraf.programming	Discussion of IRAF programming.
adass.iraf.sources	Archive newsgroup for small programs, scripts, etc.
adass.iraf.system	IRAF system issues and system administration
adass.iraf.misc	Miscellaneous discussion of IRAF software.

The easiest way to subscribe to the mailing lists for these newsgroups is via the IRAF Web pages. The URL for the ADASS newsgroups is

http://iraf.noao.edu/adass_news.html

This Web page provides links to read the news, search the news archive on *iraf.noao.edu* (all non-SPAM traffic is archived), and subscribe to the various mailing lists (you can always unsubscribe later).

The Web-hosted ADASS keyword search facility is particularly useful. For example you could review recent postings on keywords such as "xgterm", "splot", or "hpux", or find that email on preparing proceedings for the ADASS conference, which you can't find any longer in your private mailbox.

The simplest way to post to an ADASS newsgroup is via email. For example to post to *adass.iraf.applications* one could send email as follows:

mail adass-iraf-applications@iraf.noao.edu

All of the newsgroups/mailing-lists have such mail drops, although some of the newsgroups such as *announce* and *buglog* have narrowly defined purposes and may not accept all postings.

Projects wishing to create new ADASS (astronomy software) newsgroups are welcome to do so; ADASS is a standard USENET hierarchy. It might be advisable to discuss this first in *adass.admin*, or via email, to determine whether the newsgroups are appropriate for the ADASS hiearchy.

Doug Tody

IRAF Archive Mirror Sites

The IRAF FTP and WWW archives are now being mirrored on a nightly basis at several locations:

In the UK at RAL:

ftp://star.rl.ac.uk/pub/iraf
http://star-www.rl.ac.uk/iraf

In Germany at ESO ST-ECF:

ftp://ecf.hq.eso.org/iraf
http://ecf.hq.eso.org/iraf/web

In Japan at NAO/ADAC:

ftp://sinobu.mtk.nao.ac.jp/NOAO
http://sinobu.mtk.nao.ac.jp/iraf/web

Access to all mirrors is available through the IRAF homepage as well.

The IRAF Project is grateful to these sites for agreeing to host an IRAF mirror. Other sites who would like to become mirrors are encouraged to contact us for information.

Mike Fitzpatrick for
Dave Terrett, RAL
Richard Hook, ESO ST-ECF
Shin-ichi Ichikawa, NAO/ADAC

IRAF V2.11 Supported Platforms

The distribution cycle for IRAF V2.11 began with the initial release of V2.11 for the SunOS and Sun Solaris operating systems in late August 1997 (a BETA-test version was released in late June). IRAF V2.11.1 was released for the DEC Alpha (Digital Unix 4.0) and for the HP (running HP-UX 10.20) in late December; a V2.11.1 patch was released at the same time for the SunOS and Sun Solaris systems. In mid-January the first version of PC-IRAF was released, supporting Slackware Linux V3.3, Red Hat Linux V5.0, and FreeBSD V2.2.5. This release of PC-IRAF was built using IRAF V2.11.1, the same version of IRAF as released for the other platforms in late December (see more about PC-IRAF in the accompanying Newsletter article). IRAF V2.11.1 was released in mid-March for the SGI running IRIX 6.2 and the IBM Risc 6000 running AIX 4.1.

As of mid-April 1998 the only remaining V2.11 platform upgrades were for OpenVMS (for both the Alpha and VAX platforms), for DECstation Ultrix, and the two remaining PC-IRAF ports to Solaris x86 and MkLinux. The OpenVMS upgrade is nearly finished but much testing remains to be done. The Solaris x86 port has already been completed but it will be tested and released with MkLinux to save some time. We are waiting for a stable MkLinux DR3 release from Apple to go forward with the MkLinux port.

A V2.11.2 patch is also planned. So far only minor bugs have been found in V2.11. Assuming this continues to be the case and there is no urgent need for a patch, the V2.11.2 patch will not be released until we finish the first round of V2.11 platform upgrades.

As these distributions become available a message will be posted to the *adass.iraf.announce* newsgroup and mailing list. All distributions are available in the IRAF network archive at *iraf.noao.edu* in the *iraf/v211* directory. Mailed distributions may also be ordered from NOAO on various media. An order form (file */iraf/ORDERFORM*) is available in the IRAF network archive. We must charge a small cost recovery fee for mailed orders.

Support for several old platforms, such as SunOS for the f68881 and ffpa architectures, VAX Ultrix, and Mac A/UX, has been dropped with V2.11. Support for the DECstation running Ultrix will be dropped shortly; the V2.11 build will be done only if the hardware continues to function.

What about other future ports? The PC platforms are the most interesting new platforms at the moment. We plan to eventually port to Windows NT, once key components of the IRAF system software have been upgraded piecemeal to support NT, which will make the eventual IRAF port to NT a lot easier.

Doug Tody

PC-IRAF Project Status

Linux/IRAF was first released for IRAF V2.10.4 in September 1995. The Linux/IRAF port became one of the most popular IRAF platforms soon after its ini-

tial release two years ago and is now second only to SunOS/Solaris in terms of the number of sites in use. As PCs become more popular in astronomy we expect this trend will continue to grow, not only for Linux but other PC Unix systems as well.

The PC-IRAF project involves more than just a Intel Linux port however. Beginning with the IRAF V2.11 release PC-IRAF has become a multi-architecture system supporting not only the original Slackware Linux platform, but also RedHat Linux, FreeBSD and Solaris x86. A single PC-IRAF installation can support any of these platforms if the proper binaries are installed. The new Linux/IRAF versions generate ELF binaries by default, so there should be no problem linking programs (IRAF external packages and IMFORT programs) on Linux systems of recent vintage. Supporting Linux is problematic as there are a number of different distributions and, although they all share the same Linux kernel, they differ enough to be essentially different platforms. We have done the V2.11 upgrade and testing on the most current Redhat and Slackware systems since these seem to be the most popular with our users. **F2C/GCC** is used to compile all SPP and Fortran code.

FreeBSD is used at NOAO for PC-based network servers and as a development platform for IRAF, and has proven to be a very stable system, as good or better than Linux although not used as extensively in the community. Unlike Linux there is a single FreeBSD distribution. Solaris x86 is unique in that it provides the opportunity to run Solaris software and commercial development tools and compilers on PC platforms. Finally a port to MkLinux (for the Power Macintosh) is planned, which will once again make IRAF available on the Mac. By supporting all these platforms, at least initially, we're better able to compare the merits of each system.

The PC-IRAF distribution will continue to be available freely from the IRAF network archive, but we are also considering a CD-ROM distribution dedicated to PC-IRAF. This experimental CD would contain a live IRAF system allowing you to run any of the architectures and the major IRAF external packages directly from the CD-ROM without installing to local disk. For systems short on disk space such as laptop computers this could be a useful feature. The CD-ROM would also be a convenient way to obtain equivalent versions of all 5 PC-IRAF distributions without excessive download time. The CD-ROM will also contain all the IRAF documentation, including (in some cases) HTML and PostScript versions.

Not much has changed with our suggested hardware requirements for PC-IRAF. Almost any modern PC system is powerful enough to run IRAF with acceptable interactive performance. A 17" monitor with at least 2 MB video memory is still suggested as is at least 32 MB of RAM; users should consider spending a little extra in these areas to improve performance before upgrading to a faster CPU. 4 MB of VRAM would permit 24-bit truecolor display at high screen resolutions, e.g., 1280 x 1024. The minimum screen resolution recommended for running X11 and IRAF (e.g., on a laptop) is XGA, i.e., 1024 x 768.

Future possible platforms for PC-IRAF include MkLinux as mentioned above (we plan to do the port as soon as the MkLinux DR3 release is available), and looking several years into the future, probably Windows NT. There are no immediate plans to port IRAF to Linux on the Alpha, Alpha PC, or sparc systems, although as always this depends upon how much interest there is in the community.

Doug Tody, Mike Fitzpatrick

Combined Distributions for Sun/IRAF and for PC-IRAF

With the release of IRAF V2.11, the old SunOS and Solaris IRAF distributions have been replaced by a single combined distribution. This means that a single installation of Sun/IRAF on a networked server will simultaneously support IRAF sessions running on both the SunOS and Solaris operating systems.

The installation procedure is much the same as with previous individual installations of IRAF on the Suns. The most noticeable change is with magnetic tape support (*tapecap*). The IRAF system will now look for a tapecap file called "tapecap.*node*, where *node* is the hostname of the server the tapecap file

pertains to. If this file is not found then the system defaults to the generic tapecap file "tapecap". This scheme allows each node in the network to have its own individual tapecap file, providing a different namespace for each node and allowing much smaller files to be used, simplifying administration of large networks. See the article on tapecap elsewhere in this Newsletter, or the new *Sun/IRAF V2.11 Installation Guide* and *Sun/IRAF V2.11 Site Manager's Guide* for details.

In a similar way the V2.11.1 PC-IRAF release (mid-January 1998) supports the various PC operating systems as separate binary architectures within a single IRAF installation. For now these distributions include Slackware Linux V3.3, Red Hat Linux V5.0, and FreeBSD V2.2.5, with Solaris x86 and MkLinux planned for the near future. See the installation materials for PC-IRAF for details.

Doug Tody

STIS and NICMOS Calibration Pipelines

Introduction

STScI has committed itself to developing new calibration pipeline tasks in the C language (and eventually, all new software). As a consequence, the STIS and NICMOS pipelines are the first to be so developed. All STIS and NICMOS code have been written in ANSI C (including the standard ANSI C library) with no vendor extensions. The pipeline software is modular in design with dependencies on the software environment limited and managed. The pipelines use the IRAF/STSDAS/TABLES libraries to perform I/O and computational processing. The IRAF/STSDAS/TABLES routines are accessed by use of C bindings developed at STScI (which are distributed with STSDAS/TABLES 2.0). The current implementation of the bindings allows tasks to be built as IRAF native tasks or as host-level tasks (the native form of tasks can also be run at host level in the traditional IRAF manner). Currently the pipeline tasks have been built as host-level tasks because that form has been much more extensively tested.

Data from the new instruments are mapped into ANSI C data structures which correspond closely with the logical structure of the data files. The pipeline software consists of calibration steps that operate on these data structures. A major goal was to make the calibration algorithms dependent only on these data structures and not on the details of a particular file format. All I/O is encapsulated in basic routines that form a mapping between these data structures and the data files.

General Structure of Science Data

The science data files for STIS and NICMOS use FITS files with image extensions. Data from both instruments may be represented as sequences of FITS Header-Data-Units (HDUs) containing two-dimensional arrays. These HDUs are grouped as an array of science data, together with an error array and a data quality array representing, respectively, the statistical error associated with each pixel and an array of boolean conditions that identify various possible anomalous conditions that may be associated with each pixel (stored as an integer). In addition, NICMOS has two additional arrays, an array identifying the number of samples associated with each pixel and an array identifying the integration time associated with each pixel. The structure of the FITS file for both instruments is shown below. The set of image extensions that make up the associated arrays for an exposure or readout are referred to as IMSETs. The primary header contains only those keywords that apply to all the FITS extensions contained in the file. For extracted spectra, BINTABLE extensions are used where spectra are stored in individual rows using arrays in table cells.

STIS

```
Primary Header Data Unit
     general keywords
     no data
first Science HDU        \
first Error HDU           | - First IMSET
first Data Quality HDU   /
second Science HDU       \
second Error HDU          | - Second IMSET
second Data Quality HDU  /
etc.
```

NICMOS

```
Primary Header Data Unit
      general keywords
      no data
first Science HDU            \
first Error HDU               |
first Data Quality HDU        |-First IMSET
first Samples HDU             |
first Integration Times HDU  /
second Science HDU           \
second Error HDU              |
second Data Quality HDU        |-Second IMSET
second Samples HDU            |
second Integration Times HDU /
etc.
```

A number of basic ANSI C data structures have been defined to correspond to the data files described above. These correspond to various elements such as 2-d arrays of all types, individual HDUs, individual and multiple IMSETs, and whole files.

I/O Functions Related to Science Data

Supporting these data structures is an I/O interface module called HSTIO. High-level functions in the HSTIO interface include:

1) functions to initialize all data structures, to allocate and free memory needed by the data structures,

2) macros to access elements of the two-dimensional data arrays,

3) functions to get and set values associated with header keywords,

4) functions to read and write entire image extension HDUs of any type listed above.

5) functions to read and write single and multiple IMSETs.

The HSTIO interface also includes a number of lower-level functions that can be used if there is insufficient memory to store entire HDUs. These include:

1) opening and closing a particular image in a FITS file,

2) reading and writing the FITS header associated with an image,

3) reading and writing the data array associated with an image, and

4) reading and writing arbitrary lines or arbitrary two-dimensional sections of a data array associated with an image.

All of the high-level functions are implemented in terms of the lower-level functions. These lower-level functions are implemented using IRAF's image I/O, enhanced by the addition of the FITS kernel with support for image extensions.

The design of the HSTIO interface employs the principle of encapsulation: implementation details are carefully concealed within these I/O functions. There are two important consequences of this design: flexibility in usage and flexibility in implementation. First, by encapsulating all I/O operations within the HSTIO interface, the pipeline calibrations can be written in the form of pure algorithms, with few environmental dependencies. Furthermore, since these algorithms are implemented in a widely available standard language (ANSI C), they are easily used in other environments. Second, encapsulation also allows alternative implementations of the I/O functions without impacting the calibration algorithms.

CALSTIS

The Space Telescope Imaging Spectrograph (STIS) is a versatile instrument, making use of both CCD and MAMA detectors for visual and ultraviolet coverage, operating in either imaging or spectroscopic mode. It has both first order and Echelle gratings, with a wide variety of apertures ranging from large imaging apertures to long slits to short Echelle slits. Images can be full frame, binned, or a subset of the full frame. MAMA data can be taken either in accumulate or time-tag mode. This versatility has resulted in greater complexity of the pipeline processing for STIS. The computation of the expected error of each pixel of the data products is an integral part of the pipeline processing at every stage as is keeping a record of data quality for each pixel. This information is stored along with the science data as part of each IMSET described previously and is used in subsequent processing to determine which pixels should be used as part of the processing and with any

appropriate weighting factors based on the expected error.

STIS data are associated for three purposes. (1) Since the CCD detector is sensitive to cosmic rays, multiple exposures (CR-split) can be taken at the same pointing for the purpose of identifying and rejecting cosmic rays. (2) Multiple exposures (repeatobs) can be taken with either detector for time resolution or to prevent overflow. (3) For spectroscopic observations, line lamp exposures (wavecals) are taken to allow accurate location of the image on the detector. During pipeline processing, Generic Conversion combines the individual exposures into one FITS file; a second file is written for the wavecals. **Calstis** adds together the CR-split exposures prior to flat fielding, with cosmic rays excluded from the sum. Repeatobs data are maintained as separate exposures, though **calstis** does also add them together. Wavecals associated with science data are processed by **calstis** to determine the image offset from nominal, and that offset is taken into account when extracting spectra from the science data.

The basic steps performed by **calstis** are: cosmic ray rejection; bias, dark, flat, etc.; wavecal processing; 2-D spectral rectification; and 1-D spectral extraction. **Calstis** is executed in the pipeline as one program which can do all these steps. The off-line version includes this program, but the basic steps can also be executed as separate tasks. The pipeline version of **calstis** (this and the following apply to **calnic** as well) is driven by keywords in the input header, but the separate tasks take command-line switches to control the processing. These are all host level programs, but they can be run from the IRAF CL through the use of CL scripts. The script parameters include the input and output file names and calibration switches. Each script constructs a command line string, beginning with "!" and using osfn() to get the name of the executable. File names and switches are appended to the string, depending on the values of the CL parameters. Then the string is piped to the CL to execute the program.

CALNIC

NICMOS (Near Infrared Camera and Multi-Object Spectrograph) is a instrument capable of observing between 0.8 and 2.5 microns. There are three independent cameras capable of being used simultaneously; each had a different pixel size and corresponding field of view. The NICMOS data reduction and calibration pipeline is divided into two main tasks, **calnica** and **calnicb**. **Calnica** is applied to individual observations, while **calnicb** is used only for combining associated observations. **Calnica** performs the instrumental calibration of all observations and applies typical corrections such as the subtraction of detector dark current, correction for nonlinear detector response, and flat fielding. For observations obtained in the NICMOS multiple accumulate (MultiAccum) mode, in which many nondestructive readouts of the detector are performed during the course of an exposure, **calnica** applies instrumental calibrations to all readouts individually and then combines the calibrated data from all readouts into a single, final image. Cosmic ray rejection is also performed in the course of combining the images.

Calnicb processes the data from associated observations and is used only after all observations in the association have been calibrated with **calnica**. NICMOS associated observations are typically used for 1) taking multiple exposures at a single position on the sky in order to be able to perform cosmic ray rejection, 2) obtaining dithered exposures of a target in order to remove the effects of bad pixels and residual flat field errors, and 3) obtaining images of nearby off-target sky positions in order to remove the sky and telescope background signal from on-target observations. The latter is typically only of concern for NICMOS when observing at wavelengths longer than ~2.0 microns where the warm telescope and instrument optics begin to contribute a significant amount of signal.

NICMOS associated observations are a logical grouping only; the data from individual exposures within an association are stored and treated separately until they are combined by **calnicb**. The main tasks performed by **calnicb** are to perform combining operations on separate exposures—taken at identical or different but overlapping positions—and in the process eliminate cosmic rays and subtract background.

Both **calnica** and **calnicb** propagate four auxiliary data arrays associated with the science images (i.e.,

the IMSET). First, **calnica** computes statistical errors based on a combination of detector readnoise and Poisson noise in the signal. These errors are then propagated and combined with known statistical errors in all calibration data, such as uncertainties in the dark and flatfield data. Second, data quality flags, indicating known problem conditions with science image pixels (such as hot/cold pixels, saturation, etc.) are stored as bit-encoded integer arrays. These flags are used by many calibration steps to identify pixels that should not be used. Finally, two arrays are used to record the number of data samples used to compute each science image pixel value and their total exposure time. This information is computed and propagated in all steps involving image combination so that the end user will have an accurate map of the exposure time and the number of data points associated with each science image pixel.

Perry Greenfield, Phil Hodge, Howard Bushouse
Space Telescope Science Institute

It can work for Axaf Science Center (ASC) or ROSAT data.

About 10 code changes were made in various packages to fix minor problems and additional changes have been made for the implementation of ASC related tasks. Details may be found in the release notes (**help release** in the XRAY package).

To assist proposal preparation for cycle 1 of AXAF, we generated scripts to facilitate the conversion of MARX fits events files into QPOE files. We have now implemented this conversion as a task **marx2qpoe** in **xproto**. This will be available with a patch to be released during the first quarter of 1998. The patch will also include a few bug fixes, and will be announced, as usual, via our email newsletter, *Hints and Pointers*.

Dan Harris
harris@cfa.harvard.edu

PROS Report

Changes to IRAF in the V2.11 release have caused several PROS tasks to fail. Additional failures occurred because of similar effects in the TABLES package which is used extensively by PROS. Other changes are required for V2.11 such as removing the fudge-factor shift of 0.5 pixels in some QPOE tasks which were "invented" to compensate for IRAF bugs which have now been fixed. Also note that NOAO has increased the default buffer sizes for some QPOE parameters so that reading in large data files with tasks such as **rarc2pros** should have fewer failures.

These fixes have been made and the new version of PROS (PROS 2.5) was released on 10 October 1997. Our extensive testing has been done only on Sun's; we are in the process of testing the port to a DEC Alpha.

A new task **imdetect** has been added to PROS. It is located in **xproto**. It is essentially the SASS/HRI detect algorithm, but with a constant value for the background level instead of using a background map.

The NOAO Mosaic Data Handling System

A number of observatories, including NOAO, are building wide field "mosaic" cameras. These cameras produce very large digital images using the technique of placing smaller detectors as close together as possible; i.e., a mosaic of detectors. This method overcomes limitations on the physical size of current digital detectors, such as CCDs, for covering large fields, and also decreases the readout time since all the CCDs can be readout in parallel. These new cameras provide many challenges for a data handling system.

The challenges faced by a data handling system include the following. By design these wide-field cameras produce very large images. This has implications for efficient data readout, transmission, storage, display, and processing. The readout from multiple detectors needs to proceed in parallel, hence pixels from the various CCDs are interleaved, and the data handling system must "unscramble" and otherwise decode (e.g., flip) the data stream coming from

the data acquisition system. The mosaic nature of the data adds challenges in how to align, register, and combine the various pieces including dealing with the gaps between detectors.

NOAO, through the IRAF Group, has been developing a data handling system for CCD mosaic cameras. This Mosaic Data Handling System (MDHS) is used with the NOAO 8K x 8K CCD Mosaic Camera but it is not tied to this camera. The architecture of this system is open and flexible. The basic concept is that data and events are distributed between various "components" through a messaging system called a "message bus".

During a readout the data acquisition system opens a connection to the data handling system and sends a stream of messages indicating the start and end of the readout, header information, and interleaved data packets. One component of the MDHS, called the "data capture agent" (DCA), listens for this information and collects and unscrambles the data and header information to produce the output Mosaic observation file. This file, which is more accurately a type of live data structure on the message bus, is basically a collection of images from each detector (or from each amplifier if a CCD is read out with with multiple amplifiers). The data structure can be shared during or after readout with other components of the MDHS via the message bus and written to disk.

The disk format is a multi-extension FITS file. The IRAF FITS image kernel, described elsewhere in this Newsletter, allows users to directly interact with mosaic data from the MDHS. An IRAF package, called **mscred**, is being developed to process mosaic data stored in this format. **Mscred** processes the data from a collection of raw CCD images to final flux and astrometrically corrected images. The package

There are a variety of documents on the NOAO Mosaic Data Handling System which provide much greater detail than we present here. These are referenced below. They are available on the Web from the address *http://iraf.noao.edu/projects/ccdmosaic/*.

Tody and Valdes, 1998, *The NOAO Mosaic Data Handling System*	The most current and most detailed description of the MDHS based on a paper to appear in the *Proceedings of SPIE Vol. 3355*.
Valdes, 1998, *Guide to the NOAO Mosaic Data Handling Software*	A user's guide for the IRAF **mscred** package.
Valdes, 1996, *Mosaic Data Structures*	Detailed design document for the Mosaic data format and keywords.
Valdes, 1996, *Mosaic Reductions*	Initial design for the data reduction software.

The following are papers in the proceedings of the Astronomical Data Analysis Software and Systems (ADASS) conferences.

- Tody, 1997, "The Data Handling System for the NOAO Mosaic", ADASS VI, ASP Conf. Series 125, eds. G. Hunt and H. E. Payne, 451.
- Valdes, 1997, "IRAF Data Reduction Software for NOAO Mosaic", ADASS VI, ASP Conf. Series 125, eds. G. Hunt and H. E. Payne, 455.
- Valdes, 1997, "Data Format for the NOAO Mosaic", ADASS VI, ASP Conf. Series 125, eds. G. Hunt and H. E. Payne, 459.
- Tody, 1998, "Message Bus and Distributed Object Technology", ADASS VII, ASP Conf. Series 145, eds. R. Albrecht, R. N. Hook, and H. A. Bushouse, 146.
- Tody and Valdes, 1998, "The Mosaic Data Capture Agent", ADASS VII, ASP Conf. Series 145, eds. R. Albrecht, R. N. Hook, and H. A. Bushouse, 120.
- Valdes, 1998, "The IRAF Mosaic Data Reduction Package", ADASS VII, ASP Conf. Series 145, eds. R. Albrecht, R. N. Hook, and H. A. Bushouse, 53.
- Davis, 1998, "The Astrometric Properties of the NOAO Mosaic Imager", ADASS VII, ASP Conf. Series 145, eds. R. Albrecht, R. N. Hook, and H. A. Bushouse, 184.

provides tools for the important and common operation of removing the gaps in the mosaic data by taking several exposures with small shifts, called a dither sequence, and stacking the observations into a final single wide-field image with no gaps.

While operations may be performed directly on the disk format, greater efficiency is obtained by having specialized components access the data structure produced by the data capture agent through the message bus in real-time. One such component is a real-time display (RTD). The RTD displays the data in real-time doing some basic gain calibrations. The RTD includes specialized quick-look analysis features using plug-in modules. Other components, including IRAF tasks, can access and interact with the image data in the RTD.

Another MDHS component on the message bus is a data reduction agent (DRA). This component responds to events such as the end of a readout. Through a "recipe" the DRA can automatically process or reprocess data, using the latest calibration files. It is much like the traditional pipeline component of some telescope systems but it also provides a user interface (a GUI) to allow greater control by the observer over the data reduction process than is typical for a pipeline.

The MDHS is a long-term project. The most essential parts of the system described here have been implemented and have been in use with the NOAO CCD Mosaic Camera on Kitt Peak since the fall of 1997. This includes the message bus, the data capture agent, and the IRAF data reduction package, and an early version of the real-time display. The full real-time display and automated data reduction pipeline and DRA are still under development.

Doug Tody, Frank Valdes

The SAO R&D Software Suite

For those of you who may not have heard ...

In September 1997, we made a minor public release (version 1.7.1) of the SAO R&D software tree containing, among other things, a new and improved version of **SAOtng**. We invite you to retrieve the software and use it for your astronomical analysis (and other?) needs.

This autumn release of the SAO R&D package contains a handful of bug fixes to **SAOtng** 1.7. An important fix was made to support raw array files on little-endian platforms. Support also was added so that **SAOtng** will run on frame buffers deeper than 8 bits/pixel (although **SAOtng** itself uses only 8-bits of colormap). Note that this new color support still requires that the X server configuration support PseudoColor visuals—unfortunately, this appears not to be the case with most Linux X servers configured to run in 16, 24, and 32 bits/pixel mode (where TrueColor alone is supported).

SAOtng is a new version of the popular **SAOimage** display program. It is a superset of the **Ximtool** program developed at NOAO for IRAF and as such, utilizes the NOAO widget server (included in this package). It also incorporates the X Public Access mechanism to allow external processes to access and control its data, GUI functions, and algorithms. **SAOtng** supports direct display of IRAF images and FITS images (and easily can support other file formats), multiple frame buffers, region/cursor manipulation, several scale algorithms, many colormaps, and easy communication with external analysis tasks. It is highly configurable and extensible to meet the evolving needs of the astronomical community.

To add 'open software' functionality to programs such as **SAOtng**, we have developed the 'X Public Access' mechanism (XPA). Built on top of the existing Xt selection interface, XPA allows an Xt program to define points of public access through which data and commands can be exchanged with external programs. Its simple programming and command-line interface is designed so that arbitrarily large amounts of data can be transferred to and from Xt programs using XPA. Also, the data associated with a given access point can be read and written simultaneously. XPA is the precursor to a more generalized message bus system that will be built as collaboration between SAO and NOAO.

The SAO R&D software has been built at SAO on the following platforms:

Sun Sparc	Solaris 2.5	OpenWindows 3.2
Sun Sparc	SunOs 4.1.3, 4.1.4	X11R5/R6.1
HP9000/730	HP/UX B.10.20	X11R6
SGI	IRIX 5.2, 5.3, 6.2	X11R5/R6
DEC Alpha	OSF1 4.0	X11R6
Gateway P5-90	Slackware 2.0.18	X11R6.1

We develop and run the software mainly on Suns, but some time has been spent with the other (borrowed) platforms, especially 64-bit Alphas. Ports to other systems should be relatively easy at this stage. If you do a port (or try to and have problems), please let us know.

We currently are working on an update to 1.7.1 to support IRAF V2.11. Other new features in this release include support for display of FITS image extensions and FITS binary tables (e.g., X-ray event files), and support for display of catalog objects. We plan to release this upgrade on June 1, 1998.

Please visit the SAO R&D Group Home Page for news, updates, and downloads:

http://hea-www.harvard.edu/RD/

NB: Users of IRAF V2.11 should access the WWW page:

http://hea-www.harvard.edu/RD/iraf211.html

to get V2.11-enabled updates of individual 1.7.1 SAO R&D programs that will be made available before the spring upgrade.

Much of this software was developed at SAO by the HEAD Software R&D group with the significant help of collaborators across the country. The work at SAO was performed in large part under grants from NASA's Applied Information System Research Program (NAGW-3913 and NAG5-3996), with support from the AXAF Science Center (NAS8-39073). **SAOtng**, ASSIST, and XPA are embodiments of an evolving software cooperation philosophy and practice we hope to bring to astronomy and other disciplines. They reflect our efforts to understand how software systems (and people) can act in concert without sacrificing their independence.

Eric Mandel for the SAO R&D Software Group *eric@cfa.harvard.edu*

X11IRAF Project Developments

In the spring of 1997 the first "official" version (V1.0) of the X11IRAF tools (**Xgterm**, **Ximtool**, and **Xtapemon**) was released which featured many new enhancements. A second version (V1.1) was released in September. All tasks now include manual pages, and the source code can be built with a global make command on all Unix platforms currently supported by IRAF. A bug-fix and science GUI support release is planned for summer 1998.

Most of the recent work has concentrated on new features and enhancements for the **Ximtool** display server, including the following items:

- Command line options
  - More than a dozen new options for configuring **Ximtool** at startup.
- Keystroke Accelerators
  - Dozens of new keystroke commands for accessing the most commonly used **Ximtool** func tions.
- Print Option
  - Encapsulated PostScript to disk file or choice of printers
  - PseudoColor or RGB color output
  - Orientation and image scale options
  - Annotated output option
- Load Option
  - Supported image formats include FITS, IRAF OIF (.imh), GIF, Sun Rasterfile

  - Automatic grayscale conversion and z-scale transform
  - Directory browsing & filename pattern matching
- Save Option
  - Supported formats include FITS, GIF, Sun Rasterfile
  - PseudoColor or RGB output options depending on format
- Online Help
  - Builtin HTML help window
- Tcl Interactive Expert-mode Shell
- Alternative GUIs
  - Standard GUI plus image magnifier marker window
  - Alternate GUI with more command buttons in main window

The alternative GUIs provide some experimental new features which we are still working on (to improve performance, wring out the bugs and the like) and which we are not yet ready to include in the standard program. Nonetheless many users will like to have access to the new features so we are providing early access to the new features via the alternative GUIs.

The alternative GUI versions are run using the standard binary but merely execute with a modified GUI descriptor file loaded at runtime, providing an interface where some of the more commonly used features are accessible directly from the main image window as new menubars. Users can of course still customize the GUI themselves to provide a more comfortable, personalized interface without recompiling the task, either through resource settings or programming modifications to the Tcl GUI script file.

Some important issues remaining to be addressed are those of support for 24-bit frame buffers, which are common on most PC platforms, and colormap conflicts with applications such as Netscape. Work on the Gterm widget is required to fully address these issues but some workarounds are available: users running 24-bit truecolor displays must either re-start their X server in 8-bit PseudoColor mode to make use of **Ximtool**, or they can use the "Xnest" utility which allows them to run a PseudoColor visual on a 24-bit display. Colormap conflicts can be minimized by adjusting the *basePixel* and *maxColors* **Ximtool** resources (see the **Ximtool** man page for details), or preferably, by running Netscape with a private colormap (set "Netscape*installColormap: Yes" in your *.Xdefaults* file or use the "-install" command line option).

Doug Tody, Mike Fitzpatrick

Client Display Library—CDL

A new library is now available that will allow C or Fortran host programs to display a image or do overlay graphics using image servers such as **Ximtool**, **SAOtng**, and **SAOimage**. The Client Display Library (CDL) provides all the functionality found in standard IRAF tasks such as **display** and **tvmark** in a form which can be used by any host program. It may be used from IMFORT programs or can be used by any application needing to display or mark images. Features include:

Image Display:
- Connections on sockets or fifos
- Hi-level display of IRAF, FITS or arbitrary rasters of any datatype
- Cursor readback
- Subraster image I/O
- Automatic Z-scaling of images (as used in the **display** task)
- Automatic frame buffer selection option
- Hardcopy from the client

Overlay Graphics:
- Marker Primitives
  - Point (10 basic types providing dozens of combinations)
  - Line (6 styles)
  - Box (fill optional)
  - Circle (fill optional)
  - Circular Annuli
  - Polyline
  - Polygon (fill optional)
  - Ellipse (arbitrary size and rotation)
  - Elliptical Annuli (arbitrary size and rotation)
  - Text (arbitrary size and rotation)

- Allows erasure of markers

Newly added features:
- 6 line styles
- 5 font choices, include sub/superscripting and in-line changes
- SPP bindings
- ANSI C prototypes
- memory optimizations

The CDL runs on all supported Unix platforms with any of the currently used image display servers and includes full documentation and sample tasks in both C and Fortran.

The CDL and a readme is available as a compressed tar file of sources from

ftp://iraf.noao.edu/iraf/x11iraf/cdl.readme
ftp://iraf.noao.edu/iraf/x11iraf/cdl.tar.Z
ftp://iraf.noao.edu/iraf/x11iraf/cdl.tar.gz

It will also be available from the IRAF mirror sites discussed elsewhere in this Newsletter. Additional work is planned to enhance the interface and user suggestions are always welcome.

This software was developed under the Open IRAF initiative which is funded by the NASA Astrophysics Data Program (ADP).

Mike Fitzpatrick

IRAF FITS Image Kernel

IRAF has a facility called the “image kernel” which permits the high level imaging applications and the image i/o subsystem to access images of various actual types. The image kernel knows how to access an externally defined image type and provides methods for directly accessing such images within IRAF at runtime.

The IRAF FITS image kernel, now available within IRAF V2.11, allows applications that read and write images to directly access FITS image files. IRAF will read, modify, or create images in standard FITS format, operating upon either simple FITS files or individual images within multi-extension format FITS files (multiple images within the same file).

A FITS file consists of one or more FITS Units. A Unit is defined as either a Primary Header Unit (PHU or primary unit) or an Extension Header Unit (EHU or simply extension). A FITS file consists of a PHU and zero or more extensions. A FITS Unit always includes a header but may or may not include any data following the header. FITS potentially supports many different types of extensions. The FITS image kernel supports FITS multiple extension format (MEF) files but will handle only primary units or image extensions. Support for FITS tables within IRAF is provided by the TABLES external package from STScI, which supports both TABLE and BINTABLE extensions. Generic tools for handling FITS extensions within IRAF are available in the FTOOLS external package from HEASARC.

Accessing simple FITS images on disk is straightforward, and the syntax used is the same as for .imh files. But accessing MEF files is a bit more involved since there are multiple FITS images in each MEF file. The general syntax looks like

```
root.extn[extension_number][kernel_section][im_section]
```

where any or all of the bracketed fields are optional.

The “root” part of the filename is the only part required to access a FITS image. The “extn” field is the file extension with a default value of “fits”. Arbitrary FITS extensions consisting of 2 to 4 alphanumeric characters are permitted by modifying the IRAF environment variable “imextn”, as in the example below (see an accompanying article in this Newsletter for more information about “imextn” and the related “imtype” environment variable).

```
cl> show imextn
oif:imh fxf:fits,fit plf:pl qpf:qp stf:hhh,??h
cl> reset imextn="fxf:fits,fit,z0f,caf,hwf,ag,bx"
cl> flpr
```

The “extension_number” term needs to be specified if you want to access a particular image within the MEF file (if you don’t specify which image you want to access it is an error). The brackets are required, and the numbering is zero for the PHU, 1 for the first

extension, 2 for the second extension and so on. If the MEF file follows the recommended IRAF convention of a dataless PHU containing only header data and the first few extensions are images, then the images will be extensions "[1]", "[2]", "[3]", and so on.

The "kernel_section" term consists of a comma-delimited sequence of "keyword" or "keyword=value" fields used to control how the specified extension is processed. A couple of important examples are "extname" and "extver": these parameters can be used to select individual image extensions by the value of the EXTNAME and EXTVER keywords in the FITS extension headers. If a kernel section is used then it is not necessary to give the "extension_number" field to identify the image extension. For example, to list the header of an image extension with name EXTNAME='spec23' the following command could be entered:

```
cl> imheader obs72.fits[spec23]
```

The "image_section" term of the image name is the familiar IRAF image section notation used to specify the section of the image that you wish to process, e.g., "[200:400,222:300]".

The FITS image kernel supports inheritance when dealing with multi-extension FITS files. If the keyword INHERIT=T is in the header of an image extension then (by default) the PHU keywords will be combined with the image extension header keywords when the image is accessed: the header you see will be a combination of the EHU and PHU keywords.

Multi-extension FITS files can be processed by any IRAF task that already supports images, but only one image extension can be processed at a time. For example, to list the headers of several image extensions in the MEF file called 'spec' one would type

```
cl> imheader spec[1],spec[2],spec[3],spec[4]
```

or one could create a list file "spec.lis" containing

```
spec.fits[1]
spec.fits[2]
spec.fits[3]
spec.fits[4]
```

and then enter

```
cl> imheader @spec.lis
```

to list the headers of all the images listed in file "spec.lis".

Additional tools for manipulating MEF files are available in the external package **fitsutil**. This package is available in the IRAF network archive on *iraf.noao.edu* in the *iraf/extern* directory. The tasks in this package will allow the user to list all the extensions in a MEF file, delete an extension, insert an extension in the middle of an MEF file or extract one or more extensions from an input file and append them to an output MEF file.

The IRAF FITS kernel adheres to the FITS standard as stated in *A User's Guide for the Flexible Image Transport System (FITS)* from the NASA/Science Office of Standards and Technology (NOST), dated September 29, 1995. This document is accessible via FTP at *nssdca.gsfc.nasa.gov* in the subdirectory *fits*. Or you can visit the FITS Support Office Web site at *http://www.gsfc.nasa.gov/astro/fits/fits_home.html*. Another very useful source of information on FITS is *http://fits.nrao.edu/*.

Further details about the FITS image kernel and its implementation in IRAF V2.11 are given in the new *FITS Kernel User's Guide* which can be found on *iraf.noao.edu* in the file *iraf/docs/fits_userguide.ps.Z*.

Nelson Zarate

New IRAF Image Format

The native IRAF image format (known internally as OIF, or the original IRAF image format) has been modified in IRAF V2.11 to avoid file pathname length restrictions. In the process other improvements were made, and the new format is now machine independent and character data in the header is byte packed, allowing host programs to more easily be used to examine text data in image headers.

The old disk image format is still supported and is now referred to as OIF version 1 (V1). The new version introduced in IRAF V2.11 is OIF V2.

Images created with older versions of IRAF are still readable by V2.11. Images generated by V2.11 using the new format are not readable by earlier versions of IRAF. If it is necessary to generate images with V2.11 that are to be read by earlier versions of IRAF, the environment variable "oifversion" can be defined to force old format images to be written (but then there will still be pathname restrictions, etc.).

cl> **set oifversion = 1**	will write the image format prior to V2.11, V1
cl> **set oifversion = 2**	(the default) will write V2.11 images, V2

The good news is that now IRAF image headers support file path names up to 255 characters in length. Since the images are machine independent they can be shared by computers with different architectures, such as a Sun and a PC.

See an accompanying Newsletter article to read how this change affects images written using IMFORT.

Doug Tody

IMFORT Changes in V2.11

There were several changes to the IMFORT interface with the release of IRAF Version 2.11.

- IMFORT was modified to read and write the new Version 2 (V2) ".imh" image format (see the accompanying article in this Newsletter). The old format (V1) is still readable with V2.11, however. The V2 format images are machine independent, avoid long pathname restrictions, and are slightly more space efficient.

- Image headers created by IMFORT can now be any size; IMFORT image headers were fixed in size in earlier versions of IRAF.

- The "min_lenuserarea" IRAF environment variable is now supported by IMFORT. Since IMFORT is used to make host programs, IMFORT programs cannot read the IRAF version of "min_lenuserarea", so it must be defined in the host environment to be recognized by IMFORT (e.g., "setenv min_lenuserarea 64000"). The default value is 64000 chars or 800 card images.

The initial V2.11 export version of IMFORT can not write V1 images (images readable by earlier versions of IRAF), but in the V2.11.1 patch release an "oifversion" environment variable is supported by IMFORT, which allows the output version for new images to be specified. Like "min_lenuserarea" this will need to be defined at the host level to be recognized by IMFORT programs. Set "oifversion" to either 1 (for V1 images) or 2 (the new V2 images, and the default).

Doug Tody

The IMEXTN and IMTYPE Environment Variables

A new CL environment variable "imextn" has been added to IRAF with V2.11. The imextn variable associates image file name extensions, e.g., ".imh", ".fits", etc., with the external image formats supported by IRAF, allowing custom filename extensions to be assigned. A file that does not have an extension listed in either the user-defined imextn variable or in the builtin system defined default imextn will not be recognized as an image by IRAF.

The default value of imextn is as follows:

```
cl> show imextn
oif:imh fxf:fits,fit plf:pl qpf:qp stf:hhh,??h
```

Here OIF, FXF, PLF, QPF, and STF are the internal names of the runtime image formats supported by IRAF, and the valid file extensions associated with each format are listed following the colon ":". Valid internal image format names (image kernel names) are "oif" for the original IRAF format, "fxf" for the

FITS extension format, “plf” for the compressed pixel list format, “qpf” for the QPOE position ordered event list format, and “stf” for the Space Telescope GEIS image format. Additional image formats may be added in the future. The imextn string is case insensitive.

File extensions may be fixed strings or simple patterns. The “??h” in the STF kernel entry means that the STF kernel will recognize any image with a 3 letter extension ending in “h” as an STF image, unless the extension is already assigned to another kernel, for example “imh”. All kernels currently impose some limit on the length of an extension, usually 3 or 4 characters.

Not all image kernels permit file extensions to be arbitrarily assigned. The OIF, PLF, and QPF image kernels do not permit the file extension to be changed. The FXF and STF image kernels support multiple image file name extensions, and permit the user to add new extensions to the list, to change the default extension, and to remove unwanted extensions as shown below.

For example the following command will add the new extension “fts” to the file extension list for the FXF (FITS) image kernel without affecting the builtin default settings for the other image formats.

```
cl> reset imextn = “fxf:fits,fit,fts”
cl> flpr
```

After the above commands, disk files which end in “.fts” will be interpreted as FITS files although “fits” is still the default extension for the FXF kernel (because it is listed first in the imextn entry). The **flpr** command flushes the CL process cache to ensure that any imaging tasks pick up the new value of imextn. In the current system imextn is only parsed once when a process starts up. Environment variables are not intended to be used as runtime parameters, they are generally used for things that should be set once at the beginning of a session (e.g., in the *login.cl* file) and left unchanged thereafter.

The next command makes “fts” the default extension for the FXF image format.

```
cl> reset imextn = “fxf:fts,fits,fit”
cl> flpr
```

New FITS images will now be created with the extension “fts” instead of “fits”, since “fts” is the first extension assigned to the FXF format.

The environment variable “imtype” defines the default image format for new images created by IRAF. The default image type is only used if no explicit image type is specified by including an explicit image extension in the output image name. You can always force an image of a certain type to be created by explicitly giving the image extension.

The default value of imtype is “imh” which means that the default output image format is OIF or the original, native, IRAF image format. By default the following command will create an “imh” image.

```
cl> show imtype
imh
cl> mkpattern testim
cl> imheader testim.imh
```

To change the default output image type to FITS change the value of imtype as shown below.

```
cl> reset imtype = fits
cl> flpr
cl> show imtype
fits
cl> mkpattern testim
cl> imheader testim.fits
```

Including the file name extension as part of the new image name will, provided that it is one of the valid extensions listed in imextn, override the default typing mechanism. For example, the following commands will produce FITS files regardless of the setting of imtype.

```
cl> mkpattern testim.fits
cl> imheader testim.fits
cl> mkpattern testim.fit
cl> imheader testim.fit
```

Imtype can also be set to the name of the image format rather than to one of the permitted extensions, a feature that can be useful if the default image extensions change. The following two commands are equivalent

```
cl> reset imextn = oif
```

```
cl> flpr
cl> reset imextn = imh
cl> flpr
```

The imtype variable has an "inheritance" attribute which is new in IRAF V2.11. If image type inheritance is enabled, the type of a new image is the type of the parent image in a "new copy" operation. If the type of a new image is determined by inheritance the default image format specified by imtype is ignored. The default type will still be used if a new image is created which is not derived from some other image, e.g., images created by tasks such as **mkpattern** or **rfits .**

By default imtype inheritance is disabled, meaning that the value of imtype determines the type of new images to be created unless the type is explicitly specified by the image extension. The following example shows the results of inheritance in some **imcopy** operations.

```
cl> reset imtype = "fits"
cl> flpr
cl> imcopy dev$pix newpix1
cl> imheader newpix1.fits   # image type is FITS

cl> reset imtype = "fits,inherit"
cl> flpr
cl> imcopy dev$pix newpix2
cl> imheader newpix2.imh  # image type is OIF
cl> reset imtype = "fits,noinherit"
cl> flpr
cl> imcopy dev$pix newpix3
cl> imheader newpix3.fits   #  image type is FITS
```

The following examples illustrate how the image extension may be used to force the type of image to be created, regardless of the value of imtype.

```
cl> imcopy dev$pix newpix4.imh
cl> imcopy dev$pix newpix4.fits
cl> imcopy dev$pix newpix4.pl
cl> imcopy dev$pix newpix4.hhh
```

When working in mixed image type environments one should be aware that imtype is not used to determine which type of image to read or access. To be safe, when in doubt one should always explicitly include the extension in the image name. In IRAF V2.11.1 and later versions, if there are multiple images with the same root name and different extensions, an error will occur if only the root name is given.

```
cl> imdel newpix4
ERROR: Ambiguous image name (newpix4)
```

In this example two or more images exist with the same root name (newpix4) and IRAF does not know which image is being referred to, so an ambiguous image name error occurs. A command such as "imdel newpix4.fits" is necessary to delete the desired image.

Lindsey Davis, Doug Tody

New Fonts for Hardcopy Graphics

The graphics kernel IRAF uses for making hardcopy plots, the SGI kernel, was upgraded in V2.11 to use a better Roman font, and a new Greek font was added. The new Roman font replaces the old font as the default hence no special action is required to make use of the new font. The Greek font, however, must be explicitly selected with a font change directive either by the application or by the user.

One way to enable the new high-quality Greek font is to embed the "\fG" font selection escape sequence in text when you use the "T" keystroke to mark text in a plot, e.g., "\fGa \fRHydra" would switch the font to Greek, print a Greek alpha ("a"), print a space, restore the Roman font, and then print "Hydra" in the Roman font. The escape sequence "\fX" switches the text font to "X" (where "X" is 'B' for bold, 'R' for roman, 'G' for Greek, and 'I' for italic).

The greek font may also be used in label parameters for tasks like **graph**, for example

```
cl> graph dev$pix xlabel="Wavelength\\fG(A)"
```

would produce an Angstrom symbol in parenthesis. The double backslash is necessary to pass the escape string through the CL. The font may also be set explicitly using the ":.txset font=greek" command, but this will affect all text on the plot. Note that the

greek character won't appear on the screen unless the text quality is set to high, i.e., ":.txqual hi". With the default text quality setting, the hardware (e.g., X11) font is used and font selection is disabled.

A file containing the mappings for the greek fonts and other special characters can be found at *http://iraf.noao.edu/greek.ps*.

Additional work is planned in the future to further enhance IRAF graphics to provide better publication quality plots from IRAF.

Mike Fitzpatrick

IMAGES Package Reorganized

The IRAF V2.11 **images** package has been reorganized by function into the following subpackages.

imcoords	Image coordinates package
imfilter	Image filtering package
imfit	Image fitting package
imgeom	Image geometric transformation package
immatch	Image matching and combining package
imutil	Image utilities package
tv	Image display utilities package

The reorganization was driven by the need to add a large number of new tasks to the package, and to provide a structure that would make finding existing tasks simpler for the user and adding future subpackages simpler for the software developer. The new **images** package contains 82 tasks, including 26 new tasks from the **immatchx** and **vol** external addon package, 6 former **proto** package tasks, and 1 task from the **proto** subpackage of NOAO.

Major additions to images include the following:

- Several new astrometry tasks in **imcoords** (see article elsewhere in this Newsletter).
- Several new image coordinate transformation tasks, also in **imcoords** (see article elsewhere in this Newsletter).
- A new ring median filtering task plus enhancements to the existing median filter tasks in **imfilter**.
- A new 3D image transpose task in **imgeom.**
- Several new world coordinate system driven image registration tasks in **immatch** (see article elsewhere in this Newsletter).
- New intensity and psf matching tasks, also in **immatch**.
- A new task for geometrically transforming coordinate lists, plus major enhancements to the existing geometric transform operators, also in **immatch**.
- New image joining and tiling utilities in **imutil**.

The undocumented **image.imdebug** package and its 6 tasks have been removed from the **images** package. These tasks have been superseded by new tasks in the **images** or **artdata** packages that include the capabilities of the old tasks. The source for the old tasks can however still be found in the file *images$lib/zzdebug.x*, which can be compiled to recreate the old **imdebug** tasks should they be needed.

By default all the **images** subpackages are automatically loaded when the CL starts up. As a result the **images** package reorganization should be mostly transparent to the user and will not affect any existing scripts. However any **imdebug** task references will have to be changed and old parameter files will not be recognized since the package names have changed.

Lindsey Davis

New Image Astrometry Tasks

The following image astrometry tasks have been installed in the new IRAF 2.11 **imcoords** package.

ccfind	Locate reference catalog objects in images
ccxymatch	Match celestial and pixel coordinate lists
ccmap	Compute plate solutions using matched coordinate lists
ccsetwcs	Update the image coordinate system using the ccmap plate solution
cctran	Transform coordinate lists using the ccmap plate solution
starfind	Automatically detect objects in images

The main function of the new tasks is to compute plate solutions for images and store them in the image headers in a FITS compatible format. The new tasks can also be used for computing accurate pixel and world coordinates for objects in images and for determining geometric parameters for images such as the plate scale and orientation. These tasks do not comprise a fully featured astrometry package. However, they have been successfully integrated into the prototype astrometry package **tfinder** (see article in this Newsletter) and the NOAO Mosaic reduction package **mscred** (see article in this Newsletter).

The examples that follow demonstrate the basic functionality of the new tasks. To reproduce these examples on a local system copy *dev$wpix* to a local directory and edit in the missing EQUINOX keyword as shown below.

```
cl> imcopy dev$wpix wpix
cl> hedit wpix equinox 1950.0 add+
```

Get the Reference Object List

The new astrometry tasks assume that the user has a list of reference catalog objects already in hand. Reference coordinates may be obtained in many ways both inside and outside of IRAF. For example, the STSDAS **regions** task could be used to extract catalog sources appropriate for the *dev$wpix* field from the Guide Star Catalog. The **tprint** task could then be used to convert from ST binary table format to the simple text file format required by the imcoords tasks.

```
cl> type wpix.coo
13:29:47.297  47:13:37.52
13:29:37.406  47:09:09.18
13:29:38.700  47:13:36.23
13:29:55.424  47:10:05.15
13:30:01.816  47:12:58.79
```

Locate Reference Objects in the Image

The **ccfind** task locates the reference objects in the image by using either the existing image header coordinate system if any, as shown in the first example, or the image coordinate system defined by the user, as shown in the second example. In both cases **ccfind** converts the reference coordinates from the reference coordinate system (in this case J2000.0), to the image coordinate system (in this case B1950.0), uses the image coordinate system to compute approximate pixel coordinates, and then applies a centroiding algorithm to compute final pixel coordinates. The first example requires that the image header coordinate system be complete and reasonably accurate; the second example requires that the user have apriori knowledge about the coordinates of the center of the image, e.g., from the telescope, and the image scale and orientation, e.g., from the detector.

```
cl> ccfind wpix.coo wpix.match wpix usewcs+

cl> del wpix.match              # if it exists
cl> ccfind wpix.coo wpix lngref=13:27:47 \
>>> latref=47:27:14 refsystem=B1950 \
>>> xmag=-0.77 ymag=0.77
```

Compute the Plate Solution

The **ccmap** task computes the plate solution using the matched celestial and pixel coordinate list produced by ccfind and the coordinates of the tangent point, then stores the computed solution in a database file for later use by the **ccsetwcs** and **cctran** tasks. The computed solution is defined in the coordinate system of the input coordinates, and J2000.0 is

assumed by default. The task **ccmap** may be run interactively as shown below or non-interactively.

```
cl> ccmap wpix.match wpix.db lngcol=1 \
>>>  latcol=2 xcol=3 ycol=4 refpoint=user \
>>> lngref=13:27:47 latref=47:27:14 \
>>> refsystem=b1950.0
... a plot of the mapping function appears
... type ? to see the list of commands
... type x to see the xi fit residuals versus x
... type r to see the xi fit residuals versus y
... type y to see the eta fit residuals versus x
... type s to see the eta fit residuals versus y
... type g to return to the default plot
... type l to see the computed x and y scales in
    "/pixel
... type q to quit and save fit
```

By default **ccmap** performs a standard 6-coefficient linear fit to the input coordinates. Higher order fits to the residuals from the linear fit are supported by **ccmap** and the plate solution evaluation task **cctran**. However, due to the lack of an FITS convention for representing higher order plate solutions, only the linear part of the computed plate solution is normally stored in the image header by the **ccmap** and **ccsetwcs** tasks. Users running IRAF V2.11.1 or later versions can force the entire plate solution to be recorded in the image header by setting the **ccmap** projection parameter to "tnx". All IRAF tasks which compute world coordinates, e.g., **starfind**, **imexamine**, **wcsctran**, etc., understand the "tnx" system. Users planning to export their IRAF images to another image processing system should avoid using the "tnx" projection if they are concerned about portable coordinate systems.

Update the Image Coordinate System

The **ccsetwcs** task creates the image world coordinate system from the computed plate solution as shown in the example below, or from a list of parameters supplied by the user.

```
cl> ccsetwcs wpix wpix.db wpix.match
```

Compute Coordinates for Program Objects

Once the plate solution is computed and stored in the image header either the new image world coordinate system or the **cctran** task can be used to compute celestial coordinates for program objects. For 6-coefficient linear plate solutions the two methods are equivalent; for higher order plate solutions computed prior to the release of IRAF 2.11.1 the **ccmap** database record must be used to retrieve the full accuracy of the plate solution. In the following examples the **starfind** task is used to compute pixel and world coordinates for the objects that it detects in the image, and **cctran** uses the plate solution to convert the **starfind** pixel coordinates into accurate celestial coordinates.

```
cl> starfind wpix wpix.pix 1.25 100.0 wcs=world
cl> cctran wpix.pix wpix.astrom wpix.db \
>>>  wpix.match
cl> page wpix.astrom
```

Lindsey Davis

New Coordinate Transformation Tasks

Three new coordinate transformation tasks **imcctran**, **skyctran**, and **wcsctran** have been installed in the new IRAF V2.11 **imcoords** package. These tasks use the image header coordinate system to perform one or more of the following functions: 1) precess the image coordinate system (**imcctran**), 2) convert the image coordinate system from one celestial coordinate system to another (**imcctran**), 3) transform from pixel to world coordinates and vice versa (**skyctran** and **wcsctran**), or 4) locate objects detected in one image in another (**skyctran**). **Skyctran** can also be used to: 1) precess equatorial coordinates, or 2) convert coordinates from one celestial coordinate system to another. IRAF V2.11 supports equatorial, ecliptic, galactic, and supergalactic image celestial coordinate systems; earlier versions support equatorial image coordinate systems only.

The following examples demonstrate the basic functionality of the new tasks. To reproduce these examples on a local system copy the image *dev$wpix* to a local directory and edit in the missing EQUINOX keyword as shown below.

```
cl> imcopy dev$wpix wpix
```

```
cl> hedit wpix equinox 1950.0 add+
```

Precess or Transform the Image Header Coordinate System

Imcctran converts the image coordinate system from one celestial coordinate system to another. The conversion shifts and rotates the image coordinate system but leaves the pixel coordinates of the reference point and the coordinate projection unchanged. Precession is treated as a special coordinate conversion.

The following example converts the coordinate system of *wpix* from equatorial B1950.0 to equatorial J2000.0 and then from equatorial J2000.0 to galactic. The initial image copy avoids overwriting the header of *wpix* which is used in later examples.

```
cl> imcopy wpix wpix.tmp
cl> imcctran wpix.tmp j2000.0
cl> imcctran wpix.tmp galactic
```

Transform from Pixel to World Coordinates and-Vice Versa

The tasks **wcsctran** and **skyctran** convert from pixel to world coordinates and vice versa using the image coordinate system. **Wcsctran** works on images of any dimension with any valid coordinate system. The task **skyctran** works only on two-dimensional images with valid celestial coordinate systems.

The first two examples show how to use **wcsctran** to convert from pixel coordinates to world (in this case equatorial) coordinates and back again. The format and units specifications ensure that the world coordinates are written and read in hours and degrees.

```
cl>  type pix.coo
1. 1.
512. 1.
512. 512
1. 512.

cl> wcsctran pix.coo eq.coo wpix logical world  \
>>> formats="%12.3H %12.2h"
cl> wcsctran eq.coo npix.coo wpix world \
>>> logical units="h n" formats="%8.3f %8.3f"
cl> type eq.coo
cl> type npix.coo

cl> wcsctran STDIN STDOUT wpix logical \
>>>  world   formats="%12.3H %12.2h"
... type in x and y pixel coordinates and hit return
... type <EOF> to quit
cl> wcsctran STDIN STDOUT wpix world \
>>> logical units="h n" formats="%8.3f %8.3f"
... type in world coordinates in hours and degrees
    and hit return
... type <EOF> to quit
```

The next two examples show how to use **skyctran** to convert from pixel to galactic coordinates without modifying the image coordinate system which in this case is an equatorial coordinate system.

```
cl> skyctran pix.coo gal.coo wpix galactic
cl> type gal.coo

cl> display wpix 1 fi+
cl> skyctran imcursor gal2.coo wpix galactic
... point the image cursor at object on the image
    display and   type spacebar
... type <EOF> to quit
cl> type gal2.coo
```

The final example shows how **skyctran** can be used to locate objects for which only galactic coordinates are available in an image with an equatorial coordinate system. The **tvmark** task is used to mark the detected objects on the image display.

```
cl> display wpix 1 fi+
cl> skyctran gal.coo gpix.coo galactic wpix
cl> tvmark 1 gpix.coo col=204 point=15
```

Locate Objects Detected in One Image in Another

Skyctran can locate objects detected in one image in another image which has a different celestial coordinate system, e.g., equatorial and galactic as shown in the following example. **Starfind** is an imcoords task which automatically locates stellar objects in images.

```
cl> imdelete wpix.tmp   # if wpix.tmp already exists
cl> imcopy wpix wpix.tmp
cl> imcctran wpix.tmp galactic
cl> display wpix.tmp 1 fi+
cl> starfind wpix pix.wpix 1.25 100
cl> skyctran pix.wpix pix.wpix.tmp wpix \
>>>  wpix.tmp transform+
cl> tvmark 1 pix.wpix.tmp col=204
```

Precess or Convert Celestial Coordinates

The **skyctran** task is a general celestial coordinate conversion tool as well as an image coordinate conversion tool as shown in the following two examples.

```
cl> skyctran eq.coo sup.coo j2000.0 supergalactic
cl> type sup.coo
```

```
cl> skyctran STDIN STDOUT b1950.0 j2000.0
... type in b1950.0 coordinates followed by return
    to get j2000.0 coordinates
... type ":outsystem gal" to switch to galactic output
    system
... type in b1950.0 coordinates followed by return to
    get galactic coordinates
... type ":outsystem super" to switch to supergalactic
    output system
... type in b1950.0 coordinates followed by return to
    get supergalactic coordinates
... type <EOF> to quit
```

Lindsey Davis

New Image Registration Tasks

The following world coordinate system (WCS) driven image registration tasks have been added to the new IRAF V2.11 **immatch** package.

skyxymatch	Generate matched pixel lists using the image celestial wcs
skymap	Compute geometric transforms using the image celestial wcs
sregister	Register 1-D or 2-D images using the image celestial wcs
wcsxymatch	Generate matched pixel lists using the image wcs
wcsmap	Compute geometric transforms using the image wcs
wregister	Register 1-D or 2-D images using the image wcs
wcscopy	Copy the wcs from one image to another

The tasks **wregister** and **sregister** use the reference and input image coordinate systems to compute a grid of matched pixel coordinates, use the matched grid to compute a coordinate mapping from the reference image to the input images, compute the output image pixel values by interpolating in the input image at the position of the mapped reference image pixels, and update the output image coordinate systems. **wregister** can register images with any valid coordinate system provided that the reference and input image have the same coordinate system, e.g., both images have J2000.0 equatorial systems. **sregister** can only register images with celestial coordinate systems, e.g., equatorial J2000.0 or galactic, but can register images with different celestial coordinates systems, e.g., register an image with an FK4 B1950.0 coordinate system to one with an FK5 J2000.0 coordinate system.

Matched coordinate grids or coordinate mappings can be computed without doing the actual image registration by calling the supporting tasks **wcsxymatch**/**wcsmap** in the case of **wregister**, or the **skyxymatch**/ **skymap** tasks in the case of **sregister**. These tasks can be used to examine the coordinate mappings interactively.

The tasks **wcsmap**/**skymap**, **wregister**/**sregister** are script tasks which call **wcsxymatch**/**skyxymatch** to compute the respective matched coordinate grids, **geomap** to compute the required geometric transformations, **geotran** to do the actual image registration, and **wcscopy** to update the image coordinate system.

Lindsey Davis

Pixel Masks in DISPLAY, FIXPIX, and CCDPROC

We are slowly making progress on incorporating pixel masks into various IRAF tasks. A pixel mask is a special type of non-negative integer image, identified by the extension "pl" for pixel list, typically used to select a small set of pixels or regions in an associated image. A mask might be used to identify bad

pixels or regions for some operation. In IRAF V2.11 there are three tasks in the core and NOAO packages which have been revised to support pixel masks. These are **display**, **fixpix**, and **ccdproc**.

One of the common uses of pixel masks is to identify bad pixels. These "bad pixel masks" have zero values for good pixels and non-zero values for bad pixels. The bad pixel mask is assigned to one or more data images and then applications can identify good and bad pixels by matching the pixel positions in the bad pixel mask and the data image. The assignment of bad pixel masks to data images in the three tasks discussed here is done by specifying the filename of the bad pixel mask in a task parameter. As a special case the task parameter can take the value "BPM" which directs the task to get the filename of the bad pixel mask associated with an image from the keyword "BPM" in the header of the image.

The pixel positions used to match the mask pixels and the data pixels in these particular tasks use physical coordinates. For many images the physical coordinates are simply the lines and columns of the pixels in the image. In this common situation the mask and data image have the same size, pixel sampling, and alignment. However, if an image section is specified or extracted from the data image, or some operation like a block average is performed, the lines and columns will no longer match. The physical coordinates of a data pixel, however, remain the same during such operations. Thus, the masks assigned to data images will continue to be valid in these cases. If the tasks cannot match the pixel positions then a warning or error will be given.

So what do these tasks do with the bad pixels? They replace them by linear interpolation from nearby good pixels. The interpolation is either across columns or lines depending on which dimension is smallest except that fixpix also allows you to use mask values to define the direction. In **display** the pixels are replaced only in the displayed image and do not modify the data image, while in **fixpix** and **ccdproc** they modify the data image.

The **display** task also uses bad pixel or region masks in two other ways. The masks can be used to exclude pixels from the automatic scaling algorithm. Clearly bad pixels should not be used for the scaling calculation since they may greatly skew the display levels. The second use is as a color overlay. The overlay color or colors are selected by task parameters or the mask values can be used to define the colors. In an overlay zero values are transparent and only the non-zero mask values are displayed.

In the earlier versions of IRAF bad pixels could also be replaced by **fixpix** or **ccdproc**. However the bad pixels were defined by a text file of individual pixel coordinates or rectangular regions. If you still have these files and want to use them the two tasks continue to recognize them. Or you can convert such files to pixel masks using the new task **proto.text2mask**.

Frank Valdes

Changes to the RFITS and WFITS Tasks

Better support for unsigned short integer data has been added to **rfits** and **wfits**. The task **wfits** will by default transform unsigned short IRAF images into FITS images with BITPIX = 16, BSCALE = 1.0, and BZERO = 32768.0. The **rfits** task will by default transform FITS images with BITPIX = 16, BSCALE = 1.0, and BZERO = 32768.0 back into unsigned short IRAF images.

With the addition of the FITS kernel to IRAF V2.11, support for reading and writing image data to multi-extension format (MEF) FITS files has been added to the **rfits** and **wfits** tasks. Previous versions of **rfits** and **wfits** supported old-style single image FITS files only. The new **rfits** can unpack FITS image extensions into individual IRAF images, and the new **wfits** can pack individual IRAF images into a single multi-extension FITS file, as shown in the examples below.

In these examples tape files are numbered starting at one. In a MEF FITS file, the primary header unit (PHU) is FITS unit zero, and the first extension is FITS unit 1. By convention IRAF prefers to use the PHU of a multi-extension FITS file only for global header information relating to the entire file, so the

first data unit in a MEF FITS file is at index one (there is no guarantee however that all MEF FITS files will adhere to this convention). The PHU of a MEF FITS file may be read as an image regardless of whether it contains pixel data. If there is no pixel data it is an image with NAXIS=0, with an image header and no data.

1. Read all image data from all FITS files on tape into individual IRAF images.

```
cl> rfits mta "" nite1
cl> rfits mta * nite1
```

2. Read all image data from FITS files 1-8 and 10-20 on tape into individual IRAF images.

```
cl> rfits mta "1-8,10-20" nite1
```

3. Read image extensions 0-3 from FITS files 1-8 and image extensions 4-7 from FITS files 10-20 on tape into individual IRAF images. Note the use of the ranges syntax to specify both the tape file list which is one-indexed and the FITS "unit" list which is zero-indexed.

```
cl> rfits mta "1-8[0-3],10-20[4-7]" nite1
```

4. Read all the image data from a list of FITS files on disk.

```
cl> rfits @fitsfiles * nite1
```

5. Read the zeroth FITS unit (i.e., the PHU, which may or may not contain image data) from a list of FITS files on disk.

```
cl> rfits @fitsfiles "" nite1
cl> rfits @fitsfiles 0 nite1
```

6. Read the PHU and image extensions 2-4 and 6 from a list of FITS files on disk.

```
cl> rfits @fitsfiles "0,2-4,6" nite1
```

7. Write a list of IRAF images to individual FITS files on tape or disk.

```
cl> wfits @imlist mta[EOT]
cl> wfits @imlist nite1
```

8. Write a list of images to a single FITS file on tape or disk.

```
cl> wfits @imlist mta[EOT] extensions+
cl> wfits @imlist nite1 extensions+
```

The new **rfits** and **wfits** tasks and the FITS image kernel can be used to unpack multi-extension FITS image data into individual FITS images, or pack individual FITS images into multi-extension FITS files, as shown in the following examples. The equivalent **imcopy** commands which use the FITS image kernel exclusively are shown for comparison.

9. Read the third and fourth image extension from a multi-extension FITS file into an individual FITS image. Adding ".fits" to the output image name ensures the creation of a FITS image.

```
cl> rfits nite1.fits 3-4 nite1ex3.fits,nite1ex4.fits
cl> imcopy nite1.fits[3] nite1ex3.fits
cl> imcopy nite1.fits[4] nite1ex4.fits
```

10. Write a list of FITS images on disk into a single multi-extension FITS file. To maintain compatibility with the FITS image kernel, **wfits** appends the value of the "fextn" parameter to the output FITS file name.

```
cl> wfits im1.fits,im2.fits nite1 fextn=fits \
>>> extensions+
cl> imcopy im1.fits nite1.fits
cl> imcopy im2.fits nite1.fits[append]
```

To maintain compatibility with the FITS image kernel, support for global header keyword inheritance has been added to **rfits**, and minimal global header support has been added to **wfits**. The new **rfits** will automatically merge the keywords in the primary header of a multi-extension FITS file with the keywords in each image extension header provided that, the primary header has NAXIS = 0, the image extension header has INHERIT = T, and the primary header keywords are not duplicated in the image extension header. **Wfits** will by default prepend a minimal global header to any output multi-extension FITS files.

Lindsey Davis

Converting IRAF images with IMPORT/EXPORT

Two new tasks have been added to the **dataio** package for converting images to and from the internal IRAF formats:

export	Convert IRAF images to some other format
import	Convert some other format to an IRAF image

The **import** task has a user-extensible database of image formats it uses to automatically recognize and read image formats including GIF, Sun rasterfile, PGM and PPM, X11, etc. Task parameters allow for the reading of arbitrary binary files, support all datatypes and permit padding and interleaving of data. An "outbands" parameter is a list of expressions which are evaluated to compute the pixels in each band of the output image. Operands in these expressions consist of numeric constants and the pixel "tags". General arithmetic expressions are supported which can include any of the builtin functions used to change the image orientation and separate or combine input color components.

The **export** task can take one or more input IRAF images and convert them to various output formats or a raw binary raster. One of the more useful output formats supported is Encapsulated PostScript (color EPS is also supported). This task has an "outbands" expression parameter that lets you manipulate the input images, but the list of builtin functions is more extensive than with the import task. For output formats which allow color, input images may be written as individual RGB components in 24-bit formats or combined to compute an 8-bit pseudocolor colormap. Other colormaps may also be applied. Multiple input images may be combined spatially to form a mosaic of images in the output format. Images can be scaled to 8-bits using the same z-scale algorithm as in the **display** task; a brightness/contrast value may also be applied to final output image. As an example, to write four images to a grayscale EPS file, scaling each one individually such that they are tiled in a 2x2 grid:

```
cl> export im1,im2,im3,im4 quad eps \
>>> outbands="band( (zscale(i1)//zscale(i2)), \
>>> zscale((i3)//zscale(i4)) )"
```

It is possible to pad the images with blank space so they are separated with a more complex expression.

With the exception of GIF, only pixel raster formats are currently supported by either task. There is great interest in formats like TIFF and JPEG; however, these cannot be fully supported without implementing all aspects of the format (all possible tags, compression, etc.). We may add support for these formats in the future. For now it is best to use one of the supported formats as an intermediate and use host tools to convert to the final format.

Mike Fitzpatrick

DIGIPHOT Package Update

The default magnitude zero point of the **apphot** package has been changed from 26.0 to the default **daophot** package value of 25.0 in order to minimize confusion for users who switch back and forth between the two packages. For greater cross package consistency the magnitude zero point of the **imexamine** task has also been changed from 30.0 to 25.0.

The **apphot** and **daophot** package aperture photometry tasks have been modified to compute the aperture sums and aperture areas in double precision rather than in real precision. This change was made to minimize machine precision errors for sums computed over large, e.g., galaxy sized, apertures and for low noise or synthetic image data. The same tasks have been modified to output any negative total flux values should they occur, in order to make it easier to examine the effects of sky value and machine precision errors. In former versions of the **apphot** and **daophot** packages negative flux values were regarded as non-physical and were set to 0.0.

Lindsey Davis

Update on the NOAO Spectroscopy Packages

The changes in the NOAO spectroscopy packages for IRAF V2.11 are largely evolutionary. Though there are no new packages or major new tasks there are some useful new small tasks and features which are highlighted in this article.

The three new spectroscopy tasks are **autoidentify**, **skytweak**, and **telluric**. The latter two are described in a separate article. The task **autoidentify** is the result of algorithm development for automatically identifying lines in dispersion calibration spectra. The goal is to simply give the algorithm a calibration spectrum, hopefully with some approximate information about the wavelength coverage, and have it recognize the line identifications from a list of possible lines. The automatic line identification works very well in many cases but also fails in some cases where there is no information about the wavelength coverage or the dispersion relation is significantly non-linear. Additional work for future releases is being done to address the problems of non-linear dispersion functions.

The **autoidentify** task is a variant of **identify** which attempts to find a dispersion solution in the input spectrum before entering the usual interactive line identification phase. This routine is used in the various integrated image reduction tasks such as **doslit** and **dofibers**. These tasks also have new parameters for the automatic line identification algorithm such as guesses for the central wavelength and dispersion. The automatic line identification algorithm is also available as a new keystroke, 'b', in the standard interactive identify.

The dispersion units for spectra are set during the dispersion calibration process using **identify**/**reidentify**. Improvements were made to allow one to specify the dispersion units as task parameters or in a comment in the line list. Because line lists now have clearly defined units this also means one can use a line list in one set of units but do the calibrations in another set of units. The unit information is then carried along in the identify databases and in the image headers and used appropriately by all the spectroscopy tasks including the radial velocity tasks in the RV package. The end result is that you can now use the spectroscopy packages to work with data that is naturally or desired in nanometers, wavenumbers, or other units.

The other significant new feature in the one dimensional spectroscopy area is the addition of Lorentzian and Voigt profiles to the existing Gaussian profile fitting. These new profile fitting functions are found in **splot** and **fitprofs**. [Note that synthetic Lorentzian and Voigt profiles now may also be created by **artdata.mk1dspec**.]

The aperture spectrum extraction package, **apextract**, also has just a few improvements. The new version allows selecting apertures for various operations such as extraction. In other words, one may select a subset of the apertures from an aperture database to resize, recenter, or extract. The application that suggested this change was to allow sets of apertures to be extracted with different parameters.

The output extraction formats were improved for the case where each defined aperture is subdivided into a number of equal width subapertures using the "nsubaps" parameter. If the output format is "echelle" then the first subaperture from each order is written to one output Echelle file, the second to another, and so forth. The multiple output Echelle files can then be handled easily with the Echelle dispersion calibration tasks **ecidentify**/**ecreidentify**. After dispersion calibration the subapertures can then be summed back together. This technique is useful when lines of constant wavelength in each order are curved or not aligned with the image lines or columns. An output format of "onedspec" will write each subaperture to a separate spectrum file.

The last change to note in the **apextract** package is that the aperture identification information can now be specified within the image header in addition to using a separate text file. In the future multifiber or multiobject spectrographs will be able to pass this information on to the reduction software entirely within the image. For instance, the WIYN/Hydra raw data contains the information for **dohydra** to use without needing the ".iraf" file.

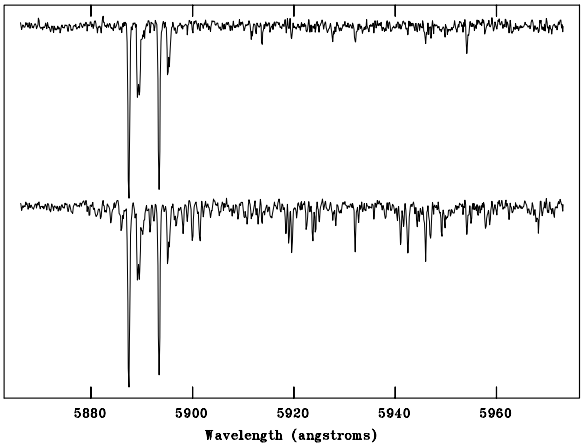
Frank Valdes

Iterative Correction of Spectra: TELLURIC and SKYTWEAK

Some types of calibration or correction operations on spectra require subtracting or dividing one spectrum by a second matching spectrum. Two examples of this are dividing an object spectrum by the spectrum of a "featureless" star spectrum to remove telluric absorption lines and subtracting a blank field spectrum from an object spectrum to remove telluric emission lines. Even though these are simple operations, despite the best efforts at data reduction, the match between the two spectra may not be sufficient to completely remove the telluric features. But the corrections can be improved by "tweaking" the spectra to improve the match and better remove the sky features. Because the features to be removed are mixed-in with features of the object spectrum this tweaking ultimately requires iteration, with the astronomer supplying the judgement as to what constitutes the best correction.

The two first order "tweaks" are small shifts in the wavelengths and finding the best intensity scale to match the spectra. One could use existing tasks to apply these tweaks, using image arithmetic tasks to perform the division or subtraction, and examining the result with something like **splot**. However this is very tedious if a lot of iterations or spectra need to be done. There are two new, related tasks in IRAF V2.11 to make this kind of iteration easier. These tasks are **telluric** and **skytweak**.

The **telluric** task intensity scales a calibration spectrum, normally that of a hot star, using a simple Beer's law to approximate the curve of growth of the telluric lines through the atmosphere. It also shifts the calibration spectrum in wavelength. The scaled and shifted calibration spectrum is divided into the object spectrum and three versions of the corrected spectrum are shown. The three versions use different values of shifting or scaling. The astronomer can then use the graphics cursor to select which of the three is best. Two new corrections with shifts or scales on either side of the selected correction are computed and graphed for the astronomer to again compare and select the best result. The program automatically or by user selection adjusts the steps between the possible shifts and scales to "home in" on the best result.



Caption: *The two spectra shown in the figure are of a giant star in M92, with and without the removal of telluric H_2O lines using the telluric task. The spectra were obtained from the NOAO WIYN Queue program. In addition to prominent stellar and interstellar components of the Na D lines, the original spectrum is cluttered with telluric water vapor lines, making the measurement of line strengths and velocities difficult. The* ***telluric*** *task was run automatically, without manual intervention to adjust the shift or strength of the telluric lines. The telluric Na D emission lines were also removed using the sky subtraction feature of* ***dohydra****, leaving spectra from which measurements can be made easily.*

There are a number of additional features to help find and evaluate the correction. An initial shift may be determined by cross-correlating the object and calibration spectra over the whole spectrum or within certain regions. Regions which are fairly clear of object features can be defined interactively or by parameters and used to have the program automatically search for the scaling and shift that minimizes the RMS of the calibrated spectrum in those regions. The RMS is also shown during the interactive iterations as a guide to how well the correction is being done. The original object and calibration spectrum can also be graphed to help the user decide which features may or may not be removed.

The **skytweak** task is basically the same program except that the scaling is simple multiplicative scal-

ing and the corrected result is the subtraction of the sky spectrum from the object spectrum. The same features of cross-correlations, pointing to the best result of three, and automatic RMS minimization in certain regions are available.

Learning to use these tasks takes some practice, but once you understand the features and how to approach the desired result they can be extremely powerful and quick to use to improve your telluric or night sky corrections.

Frank Valdes

IMTYPE and the IMRED Spectral Reductions Tasks

As discussed elsewhere in this Newsletter there have been some important changes concerning use of different image disk formats. The goal is to allow you to select the types of image formats and extensions to be used by IRAF image tasks and to allow you to use image names without an explicit extension. These features create additional challenges for the software.

The major spectral reduction tasks in the **imred** spectroscopy packages—**dofibers**, **doslit**, **dohydra**, **doecslit**, **doargus**, and **dofoe**—are not fully general in supporting all combinations of image formats and image kernel variable specifications. Further system development is planned to allow scripts to deal with the variety of image types and extensions. If you do nothing at all and continue to use the defaults of ".imh" images then these useful tasks will continue to work as before; this is the most thoroughly tested mode of operation. However if you want to use other image types then this article summarizes what you need to do. Don't panic—you *are* able to use your favorite image disk format.

The requirement is that you must use the same image type and extension for the input and output, and the extension must be the default for the image type. Start by identifying the image type and extension for your input data; it must all be of the same type. For example, your raw data may be OIF with extension ".imh", STF format with extension ".c0h", or FITS format with extension ".fits". Now look at the **imextn** variable:

```
cl> show imextn
oif:imh fxf:fits,fit plf:pl qpf:qp stf:hhh,??h
```

The requirement means that for a particular format the first extension must be that of your input data. From the above example the default is fine for OIF and FITS, but the STF definition would need to be changed to "stf:c0h,??h". If instead of the default ".fits" extension you wanted to use ".fit" you would need to change the FXF part to "fxf:fit,fits".

Now you have to tell the reduction scripts the extension to expect. This is done with the "imtype" variable. Nominally the imtype variable would specify the image type and have some other qualifiers. The reduction scripts require that the first part of the specification be the actual extension followed by the optional qualifiers. So, if you have extension ".c0h" for STF format data you would need to set imtype and imextn as below.

```
cl> reset imtype=c0h[,inherit|noinherit]
cl> reset imextn=stf:c0h[<other stuff>]
cl> flpr    # To make change take effect
```

The items in the brackets are optional. To understand what is going on, the reductions scripts use the part before the first comma as the extension for the input data.

To conclude let us consider the three most likely cases. If your input data has the ".imh" extension then you should have the following:

```
# Acceptable values for .imh input.
    cl> show imextn imtype
    oif:imh fxf:fits,fit plf:pl qpf:qp stf:hhh,??h
    imh
```

These are the defaults provided if you don't change anything in your login files. If your input data has the ".fits" extension then you only need to set imtype to "fits" as shown below.

```
# Acceptable values for .fits input.
    cl> reset imtype=fits
    cl> show imextn imtype
    oif:imh fxf:fits,fit plf:pl qpf:qp stf:hhh,??h
```

```
fits
cl> flpr
```

Finally if you want to use ".fit" as the extension then you need to set both "imtype" and "imextn". The following is one way to do it.

```
# Acceptable values for .fit input.
cl> reset imtype=fit
cl> reset imextn=fxf:fit
cl> show imextn imtype
fxf:fit
fit
cl> flpr
```

The imtype variable can include a following ",inherit" or ",noinherit" and you can include fields for other image types or FITS extensions in the imextn variable.

Frank Valdes

Hints on Reading and Writing FITS Files

IRAF V2.11 includes the new FITS image kernel, as mentioned in other articles in this Newsletter. The **dataio** tasks **rfits** and **wfits** have been enhanced in V2.11 to support reading and writing multi-extension FITS files (FITS files containing more than one image). The focus of this article is to alert the IRAF user to differences between V2.11 and earlier versions of IRAF in handling single-image FITS files, in particular, and image files in general. A general rule to keep in mind is that IRAF V2.11 requires that all image files have some type of file name extension to define which type of image they are, be they native IRAF images, FITS images, ST GEIS-format images, and so on (the one exception to this rule is **rfits**, which will read FITS format disk files that do not have any extensions).

The syntax for reading single-image FITS files from tape has not changed. In the following example we've set the "oldiraf" parameter to "yes" to attempt to restore the original IRAF names:

```
cl> rfits mtf 1-3 image old+
```

But what if you would like to read these same files onto disk, not as IRAF images, but as FITS files? You will then need to tell IRAF that you wish to generate FITS images on disk; this can be done by setting the "imtype" environment variable to an appropriate value (see more on imtype and the new environment variable "imextn" elsewhere in this Newsletter).

```
cl> set imtype=fits
cl> flpr                          # important!
cl> rfits mtf 1-3 image old+
```

For single image tape files, this will create images "image0001.fits", "image0002.fits", etc. It should be noted, however, that these images have been processed by the FITS kernel and that their image headers will thus be modified slightly in the process. Using **rfits** in this way should be fine for most single image files, but multi-extension FITS files would be split apart by this command. When a direct copy of FITS files (multi-extension or not) from tape to disk is desired, the **reblock** or **t2d** tasks may be used:

```
cl> reblock mtf image0 1-3
cl> t2d mtf image0 1-3
```

This creates FITS files "image0001", "image0002", and "image0003" on disk. If these disk files are to be used by any IRAF applications (other than **rfits**) extensions must be added to these filenames. This can be done with the **rename** task:

```
cl> rename image0* fits field=extn
```

Now the images will be called "image0001.fits", and so on.

If you have FITS files on a tar tape (written prior to IRAF V2.11), then when these FITS files are restored to disk they will probably not have any file name extensions associated with them. If you plan to use these files directly with IRAF applications you will need to use **rename**, as above, to add the ".fits" extensions to these files before IRAF applications will recognize them as image files.

Restoring single-image FITS files on disk to IRAF images is done with **rfits** as before. But many users may find that their favorite **rfits** command will no

longer work as expected. To read FITS files on disk to IRAF images use a command similar to

```
cl> rfits fits* 0 image old+
```

Note the use of "0" for the "File/extensions list" parameter. If you are used to using "1" for this parameter value, you will need to change your ways. Now that **rfits** recognizes multi-extension FITS files, "0" refers to the primary (first) FITS unit and "1" refers to the first FITS extension in a multi-extension FITS file. For a classic FITS image with no extensions, the image data is in the first FITS unit, number 0.

Writing IRAF images to a FITS-formatted tape is done with **wfits** as in previous IRAF versions. Note however that if you use **wfits** to generate FITS files on disk that, by default, these FITS images will have a ".fits" extension. See the parameter file for **wfits**.

If you are writing single or multi-extension FITS files from disk to tape then there are several considerations.

To directly write disk FITS images to tape, **reblock** may again be used.

```
cl> reblock image0*.fits mtf newtape+ \
 >>> outb=28800
```

Here we have specified that we are writing to a blank tape, and have set the output block size to 28800 bytes (ten 2880 FITS logical records per block).

The **wfits** task could also be used to copy files from disk to tape. By default each disk image would become one tape file

```
cl> wfits image0* mtf newtape+
```

Or one may also use **wfits** to combine individual disk images into a multi-extension FITS file on tape, with or without a global FITS header:

```
cl> wfits image0* mtf newtape+ extensions+ \
>>> global_hdr-
```

In this example, no global header would be created, and so the first file on disk would become the zeroth extension in the tape FITS file.

Dave Bell

The TAPECAP File Explained

Since the *tapecap* file was introduced in V2.10 some of the most common technical support questions have dealt with how to add a new device to this file, and the meaning of the various fields in each entry. It is therefore worth spending a little time explaining what this file does and how it can be configured.

In IRAF V2.10 there was one tapecap file per IRAF installation and all client nodes sharing the same central installation required device entries in the global tapecap file. In V2.11 this scheme has been generalized to allow each host to have its own private tapecap file, with a fallback to the generic tapecap file if no host-specific file is found. The system will look first for a configuration file called "tapecap.*node*" where *node* is the hostname of the server the tapecap file describes. If this file is not found the default tapecap file, *dev$tapecap*, will be used.

Since magnetic tape (magtape) devices are interfaced differently in SunOS and Solaris but a single IRAF installation now supports both, the distributed Sun/IRAF system comes with separate versions of the generic tapecap, one for SunOS and one for Solaris (a similar setup exists for PC-IRAF as well):

```
tapecap.sunos     # SunOS template
tapecap.solaris   # Solaris template
```

The actual tapecap file for a client node can be set up (on Unix systems) as a symbolic link to a locally customized shared version of one of these files, or it could be a separate file which has been simplified and customized for a particular host. The latter approach eliminates device name conflicts and allows simple device names such as "mta" and "mtb" to be used for each node. For example, if we have hosts "corona" and "dosequis", the primary magtape device on each node could be named "corona!mta" and "dosequis!mta" (i.e., device "mta" on both

nodes) and these devices would be accessible from any IRAF host on the local network.

As an example, assume we have a heterogeneous network of systems which are predominantly Solaris workstations all sharing a single central IRAF V2.11 installation. The tapecap files in the *dev* directory might then look something like the following:

```
link    tapecap -> tapecap.solaris
file    tapecap.sunos
file    tapecap.solaris
file    tapecap.corona
file    tapecap.dosequis
link    tapecap.ursa -> tapecap.solaris
link    tapecap.gemini -> tapecap.sunos
link    tapecap.tucana -> tapecap.sunos
(etc.)
```

In this example some of the client nodes have private tapecap files, while others share a central generic OS-dependent tapecap file. The catch-all default entry "tapecap" defaults to the generic tapecap file for Solaris.

The tapecap files included in the distributed system include some generic device entries for both SunOS and Solaris such as "mtxb1" (Exabyte unit 1, Sun ST driver), "mthp2" (HP7880 9 track drive, unit 2), and so on, which you may be able to use as-is to access your local magtape devices. Most likely you will want to add some device aliases, and you may need to prepare custom device entries for local devices. There must be an entry in the tapecap file for a magtape device in order to be able to access the device from within IRAF. All magtape device names must begin with the two-letter prefix "mt".

Configuring New TAPECAP Entries

The tapecap file is a text data base file (similar to the termcap and graphcap files and adhering to the standard BSD Unix *termcap* file format) describing the capabilities and device names associated with a particular tape device on the system. For information on the format of the file see the Unix *termcap(5)* man page. A listing of all recognized device fields is given in the program comments for the tape driver in *iraf$unix/os/zfiomt.c* (more on this later). In general creating a new tapecap entry for a device is a matter of finding a similar entry in the distributed file, and either using that directly if the device names are correct, or simply modifying it slightly to change device names so it will be appropriate for a drive on a different SCSI unit. On occasion other tapecap parameters will need to be added to correct for specific behavior that affects appending new data and tape positioning.

By convention a tapecap entry for a device is usually divided into three different sections: a high-level entry giving the host-specific name of the drive as known to IRAF, a mid-level section defining the host device names associated with the drive, and a low-level entry defining generic capabilities associated with all instances of a particular type of drive (DAT, Exabyte, 9-track, etc.). The starting point for the tapecap entry is whatever IRAF name was used to access the drive. This is usually something like "mta", "mtb", etc., but can be any valid name beginning with an "mt" prefix and which defines all the needed parameters. When searching for a particular tapecap parameter the first occurrence of that parameter in the entry is used by the system, and a complete tapecap description is composed of all the entries which are linked by the ":tc" continuation fields.

As an example consider a typical entry for a DAT drive on unit 0 known to a Solaris/IRAF system as "mta", the high-level entry would look like:

```
mta|Generic DAT entry, unit 0|  :tc=mtst0.solaris.dat:
```

Here we define the IRAF name (which must begin with an "mt" prefix) along with any aliases delimited by the "|". The ":tc" field indicates that the tapecap entry continues the next entry named "mtst0.solaris.dat". This is an alias for the "mtsd0" entry as shown below:

```
mtsd0|mtst0.solaris.dat|DAT drive on Solaris:\
  :al=0 0bn 0cb 0cn 0hb 0hn 0lb 0ln 0mb 0mn 0u 0ubn \
   0b 0c 0cbn 0h 0hbn 0l 0lbn 0m 0mbn 0n 0ub 0un:\
  :dv=0bn:lk=0:tc=solaris-dat:
```

This entry is primarily used to specify the host device names associated with the drive. The ":al" (aliases) field is a list of all device aliases in the Unix */dev* or */dev/rmt* directories associated with this device. This information is needed so the tape allocation task can change the permissions and ownership on each device name which accesses that tape drive. The

":dv" (device) field is the no-rewind device name and is the device file actually opened for tape I/O. This must be a no-rewind device for IRAF to be able to keep track of the tape position. The actual device name typically depends on the density of the tape, whether compression is used, etc. The ":lk" is used to build the name of a "lok file" that is created in the */tmp* directory of the machine hosting the drive that will be used to maintain the tape status and position information; this value should be unique for each drive on the machine to avoid conflicts.

When configuring a new tapecap entry, all one usually needs to change is the IRAF device name in the first section and the host device names in the ":dv", ":al" and ":lk" fields of this entry. The host device name entry continues with a ":tc" field saying to branch to the "solaris-dat" generic device entry:

```
solaris-dat|sdat-60m|Sun/Solaris DAT drive:\
:dt=Archive Python 4mm Helical Scan tape drive:tt=DG-60M:\
:ts#1274378:bs#0:mr#0:or#65536:fb#10:fs#127000:mf:fe#2000:
```

The low-level entry here is where parameters relating to all drives of a particular type using a particular host tape driver are maintained, e.g., the record sized used for tape I/O, positioning capabilities, filemark sizes, etc. These will rarely need to be changed from the distributed entries unless you are using a new tape driver or a different model tape drive, or a type of tape cartridge with a capacity different than that given ("tz"). See the section below for a full list of the tapecap parameters and their meanings.

For a more complicated example let's consider how to add an entry for an Exabyte 8505 drive given an existing entry for an Exabyte 8200 device. We can ignore for now the low-level entry found in the distributed tapecap and concentrate on what fields actually need changing in this case. We begin with the high-level entry defining the IRAF names, we will need one name for the drive in each of three modes (8200 mode, 8500 mode, and 8500 mode with compression):

```
mta|Exabyte 8200, Unit 0|          :tc=mtst0.solaris.exb8200
mtb|mtblo|Exabyte 8505, Unit 0|    :tc=mtst0.exb8505-lo:
mtbhi|Exabyte 8505, Unit 0|        :tc=mtst0.exb8505-hi:
mtbc|Exabyte 8505, Unit 0|         :tc=mtst0.exb8505-c:
```

The new IRAF names are therefore *mtb* (8200 mode), *mtbhi* (8500 mode), and *mtbc* (8500 + compression). These all link to the second level entry where we make use of the existing EXB8200 entry:

```
mtsee0|mtst0.solaris.exb8200|Exabyte 8200 drive on Solaris:\
  :al=0 0bn 0cb 0cn 0hb 0hn 0lb 0ln 0mb 0mn 0u 0ubn \
  0b 0c 0cbn 0h 0hbn 0l 0lbn 0m 0mbn 0n 0ub 0un:
  :dv=0bn:lk=0:tc=solaris-exb8200:
mtsee0lo|mtst0.exb8505-lo|:dv=0lbn:tc=mtsee0:
mtsee0hi|mtst0.exb8505hi|:dv=0mbn:fs#48000:ts#5000000:tc=mtsee0:
mtsee0hic|mtst0.exb8505c|:dv=0cbn:fs#48000:ts#5000000:tc=mtsee0:
```

Note that the names we just created link to the one-line entries below the standard EXB 8200 entry "mtst0.solaris.exb8200" (the mtb entry could just as legally have linked to this entry right away). Since all we need to change is the ":dv" field (because we're opening the same drive, but by using a different name the host system accesses it in the appropriate mode) we can simply make a new entry point, change the ":dv" field and then link to the existing entry where all the rest of the parameters will be the same. In this case we've also reset the ":fs" and ":ts" fields to override the values in the low-level Exabyte description since these have also changed for the new model drive. If we wished to modify this entry for a drive on, e.g., unit 2, all we would need to do is modify the various ":dv", ":al", and ":lk" fields so the device names are correct, and change the name of the tapecap entry points so we avoid any confusion later on.

When configuring a new tapecap and encountering problems it is useful to turn on status output so you get a better idea of where the tape is positioned and what's going on; to do this use the ":so" (status output) field. This can be included directly in the tapecap entry for the device or temporarily enabled on the command line as in the following examples:

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

Alternatively, the :so can be specified on the command line, e.g.,

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

Any other tapecap parameters can be included on the command line in the same way; these will override

the defaults given in the tapecap file. The quotes around the tape name are required if any special characters such as "=" are included in the device name string. Status output like this can also be directed to an **Xtapemon** tape status server running either locally or remotely; see the **Xtapemon** man page for details. Help with configuring new tapecap entries is available from IRAF site support.

More on TAPECAP Parameters

As we see from the previous section, in most cases the only tapecap parameters that need to be changed are ":dv", ":al", and maybe ":lk". There are however a number of other tapecap parameters that sometimes must be modified to describe how the tape device operates or to optimize I/O to the device. A full listing of the available tapecap parameters can be found in the program comments for the IRAF tape driver (file *iraf$unix/os/zfiomt.c*); we will only briefly discuss a few here. Any changes you make with the parameters mentioned here can usually go in the low-level tapecap entry so they will "fix" all drives of the same type. However, you may prefer to modify just the high-level entry so that only one drive is affected, particularly if you are correcting a feature that is only seen on a particular host and which does not reflect the generic behavior of the device. For example:

```
mta|Generic DAT entry, unit 0| \
    :se:ow:tc=mtst0.solaris.dat:
```

would add the ":se:ow" fields (discussed below) to only the mta device.

Boolean tapecap parameters may be negated if you are linking to an existing entry which already defines a particular field. For example, in

```
mta|Generic DAT entry, unit 0| \
    :se@:tc=mtst0.solaris.dat:
```

the "@" character would negate the ":se" field regardless of whether it is defined elsewhere in the entry.

One of the most common problems encountered is that only odd-numbered images on a tape are readable by the drive. The solution to this is usually to add a ":se" to the tapecap to tell the driver that the tape will position past EOT in a read. Another common problem is with appending new data to an existing tape; this sometimes requires the addition of an ":ow" field to tell the driver to backspace and overwrite the EOT when appending. A ":re" is sometimes needed if there is a problem sensing the EOT when reading all images from a tape; this tells the driver that a read at EOT returns an ERR.

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 V2.11 **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, e.g., most Exabyte drives, 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 for some devices in the time required to execute a tape scan of a tape containing many small files.

On a device such as most cartridge tape devices where backspacing is not permitted or does not work reliably, it may be necessary to set the ":nf" parameter to tell the driver to rewind and space forward when backspacing to a file.

Lastly, when configuring a new low-level generic entry for the device it is sometimes necessary to

change the various size parameters for the drive. These include:

bs	device block size (0 if variable)
fb	default FITS blocking factor (rec-size=fb*2880)
fe	time to FSF equivalent in file Kb
mr	maximum record size
or	optimum record size
fs	approximate filemark size (bytes)
ts	tape capacity (MB)
dn	density

All but the last three fields are used either by the driver or a task when reading or writing a tape. The ":fs", ":ts" and ":dn" fields are used by tape monitoring tasks such as **Xtapemon** to compute the approximate amount of tape used and do not affect tape operation. For devices which are capable of variable block size I/O (i.e., almost anything but a cartridge tape) it is best to leave the ":bs" field set to zero. The maximum and optimum record sizes, the ":mr" and ":or" fields, are usually determined by the host tape driver used. Values for these can either be found in the host driver man page or its system include file.

Mike Fitzpatrick, Doug Tody

Measuring Coordinates with the FINDER Package

The IRAF external package, **finder**, provides tools for measuring celestial coordinates from a CCD or other digital image. A plate solution of the HST Guide Star Catalog sources in the field can be transferred to a list of program objects specified by the user. A typical use for the package is to measure coordinates in preparation for observing with a multi-object spectrograph.

The help menu will provide an idea of the package's capabilities:

catpars	Catalog column description pset
cdrfits	Read FITS tables from the GSC CDroms
disppars	Image display parameters for **tfinder** and **dssfinder**
dssfinder	**Tfinder** tailored for Digital Sky Survey images
finderlog	Generate log file from **tfinder** format table
gscfind	Search the GSC index for sources within a field
hydrahints	Phil's hints for Hydra users
mkgscindex	Regenerate machine dependent GSC CD index from FITS index
mkgsctab	Make a pseudo-GSC coordinate table from a text coord list
mkobjtab	Convert objects in input table to catalog sources in output
objlist	Print a text list of the object coordinates
selectpars	Source selection criteria pset
tastrom	Script frontend for **astrom** foreign task
tfield	Extract sources within a field from a list of tables
tfinder	Search the catalog—predict, center and fit the coords
tpeak	Interactively center table sources (called by **tfinder**)
tpltsol	Perform plate solution from measured Guide Stars using **ccmap**

Typically, a user will only need to interact with the **tfinder** task, which is a script that handles the various steps from retrieving basic field information from the image header, through searching the GSC index, retrieving catalog entries overlying the field, predicting X,Y coordinates and entering an interactive source centering and fitting routine driven by the image display cursor. The plate solution can be restricted to subsets of the GSC sources based on the plate ID, magnitude range, or any other field from the catalog. A plate solution can also be transferred to a secondary grid of non-catalog objects that can then themselves be used to fit a solution in a second (typically more deeply exposed) image.

The **finder** package relies on the IRAF V2.11 **imcoords** package tasks **ccmap** and **cctran** to perform

the interactive plate solution. Alternately, a script **tastrom** is provided that will pass the centered catalog coordinates from **tfinder** to the popular Starlink **astrom** plate solver. The GSC searching and retrieval functions of **finder** derive from the STSDAS package, **gasp**. The output of **finder** is an IRAF table containing the centered catalog and program objects, as well as a **cctran** format database containing the fitted plate solution.

The **finder** package is an interim facility pending the completion of the full IRAF astrometry package, which will address many additional astrometric chores. An earlier version of **finder** has been used for over five years, however, with great success. The current version has some restrictions, most of which can be worked around. The two main restrictions are that no direct support is provided for catalogs other than the HST GSC (the **mkgsctab** task allows output from other catalogs to be reformatted as input for **finder**, however). Secondly, no support is provided for making proper motion corrections, but these could be applied on a field by field basis to the input catalog using simple table operators.

Finder is available as *ftp://iraf.noao.edu/iraf/extern/finder.tar.Z*. Downloading and installation instructions are in the file *finder.readm*e in the same FTP directory.

Rob Seaman

The Mosaic Data Reduction Package—MSCRED

NOAO and other observatories have been developing and operating cameras consisting of mosaics of CCD detectors. The NOAO Mosaic Camera consists of 8 CCDs producing an 8K x 8K format. The IRAF group has been designing and developing software to handle data from these types of cameras. The first version of the data reduction software within the V2.11.1 IRAF environment is now available. It consists of an external IRAF package called **mscred** which provides tools for CCD processing, mosaic display and examination, mosaic reconstruction, and combining dithered observations. Future developments call for pipeline processing, increased support for bad pixel masks, and the determination and propagation of uncertainty information. This article briefly highlights the features of this first version of **mscred**.

The **mscred** package operates on data from a mosaic of CCD detectors recorded as a set of CCD images, one from each CCD amplifier in use, stored in a multi-extension FITS (MEF) file. This is a convenient format since it keeps all the data from a single observation in a single file and users need only specify one filename to operate on all the CCD data from the mosaic camera.

The tasks in the **mscred** package perform the following functions:

- display the mosaic data as an apparent single mosaic image
- provide interactive examination of displayed mosaic data
- combine multiple calibration exposures into master calibration files
- perform the basic CCD calibrations such as zero level and gain
- reconstruct a single mosaic image with distortions removed
- register and combine dithered operations
- save and restore data on tape

The display tools in the first version of the **mscred** package use an interim approach based on the standard display servers such as **Ximtool**. Another part of the on-going software development effort for the Mosaic Data Handling System involves a full-featured real-time display. The interim approach maps each piece of a mosaic observation into a part of a standard 8-bit display buffer using a display “fill” operation. Thus the display provides a visual approximation of the full data as well as approximate coordinate and pixel values in the WCS box of the display server. The user can interact with the full image with a version of **imexamine**. The image display cursor selects an operation and position in the mosaic observation that is used to access the mosaic observation on disk at full spatial and intensity resolution. Similar tasks allow measurement of the PSF and focus in the full observation.

The basic calibration processing of the data is essentially the same as that used for single readout CCD images. In fact the task names and parameters are almost identical with those in the **ccdred** package. So standard processing consists of using **zerocombine** to combine zero time exposures, **darkcombine** to combine dark count exposures, and **flatcombine** to combine flat exposures, and **ccdproc** to process the data exposures using the combined zero level, dark count, and flat field calibrations. What actually happens is that the standard **ccdred** operations are performed on each of the matching mosaic pieces.

The mosaic and dither reconstruction tools are based on the world coordinate system (WCS). Each amplifier image of a mosaic observation has an independent WCS. For the NOAO Mosaic Camera the WCS for each element of the mosaic was calibrated from observations of astrometry fields. This provides accurate celestial coordinates apart from a zero point. When a observation is recorded the WCS calibration is included in each element of the mosaic and the zero point is set to the telescope pointing. Tools are provided to adjust the zero points interactively from a display and to register dithered observations to a common zero point.

The mosaic reconstruction task creates an empty image with a single simple undistorted WCS that is just large enough to include all the pieces of the mosaic observation. Each mosaic piece is then resampled into the empty image using a mapping from the piece's WCS to the final image WCS. For dithered observations the WCS for each mosaic reconstructed image are also adjusted so that they can be combined by simple integer shifts. The combining is done excluding the gaps and bad pixels from each observation. This ends up producing a final image that has the gaps in the mosaic filled in, bad pixels removed, and a WCS that gives accurate coordinates.

Information about the NOAO Mosaic Project can be found at the following URL: *http://www.noao.edu/kpno/mosaic/mosaic.html*. It includes links to design documents for the data handling system, the data format, the **mscred** package, and a *User's Guide to the MSCRED Package*.

Frank Valdes

Archiving Data with "Save The Bits"

The NOAO/IRAF "Save The Bits" archive (STB) has been in operation on Kitt Peak since July 1993, automatically archiving newly acquired KPNO and NSO nighttime optical and IR digital images from eight different telescopes. A duplicate STB archive was installed at CTIO in April, 1996, saving data from four additional NOAO telescopes. Over 1.5 million images have been archived to date, amounting to approximately 3.5 Terabytes of scientific imagery. This total is currently increasing at over one Terabyte/year.

This rather large rate of astronomical data production is likely to more than double with the addition of data from the BTC Mosaic camera (with four 2K x 2K CCDs) at CTIO last Fall, and from the NOAO Mosaic (with eight 4K x 2K CCDs) at KPNO in February 1998. Archiving the NOAO Mosaic images (large multi-extension FITS files at 135 MB each) required a port of STB to Solaris and is accomplished with dedicated Exabyte 8705 drives attached directly to the Sun UltraSparc used by the Mosaic Data Handling System. The smaller (but still impressive) BTC data stream was simply attached to the general CTIO STB installation.

Other observatories that use STB include UCO/Lick and the W. M. Keck Observatory, where STB has been in continuous service since February, 1995.

Each of the original NOAO STB installations consists of four Exabyte 8505 tape drives arranged as two pairs of duplicate tape copies. One copy of each tape is shipped to the central NOAO data center in Tucson, while the second copy is retained on site at each observatory. The data from all of the telescopes on each mountaintop are transported via the local Ethernet to a central archive computer where the images from the several telescopes and instruments that are in operation on any given night are interleaved onto the same media and are packed into large FITS image extension files to optimize efficient access for later retrieval.

STB uses the Berkeley Unix **lpd** print spooler to provide the underlying network queuing mechanism. Queuing via **lpd** has been used with robust results at NOAO for many years to simplify file transfers to other facilities such as photographic hardcopy units. In the simplest case, the data acquisition software for an astronomical instrument writes a FITS file to disk on a computer in the telescope dome. This FITS file is then simply passed to a standard network **lpd** queue, "bits", which passes the file to the queue on the central archive computer.

On the archive computer, the bits **lpd** queue passes each FITS file in turn to the actual STB archive daemon, "bitf". The bitf daemon is responsible for adding a serial numbered keyword to each FITS header and for translating simple FITS images into the FITS image extension format. A FITS checksum (*ftp://iraf.noao.edu/misc/checksum/checksum.ps*) is calculated for each FITS extension to support later tape verification. After a sufficient volume of data (approximately 50 Mbytes) accumulates, the individual FITS extensions are assembled into a complete FITS multi-extension file on tape. The duplicate tape copies are written at the same time by interleaving writes from the same data buffer to separate tape drives.

After a pair of tapes are full, the first duplicate set of Exabyte drives are rewound and verified against the checksums, as well as against each other, while the second pair continues with taping duties. The capacity of an Exabyte 8505 format tape is conservatively rated at 4.12 Gbytes (this corresponds to a round number of 1.5 million FITS records of 2880 bytes each). When empty tapes have been newly mounted in both pairs of duplicate drives, and allowing for 2 Gbytes of queue spooling area on the archive computer, the total capacity of the online media is greater than 10 Gbytes. Note that if both pairs of tapes are allowed to fill up without swapping the media, and after the spool area fills up on the archive computer, that **lpd** provides its normal service of continuing to queue the data onto disk on the data acquisition computer in each dome. After the tapes are swapped (or after some network or computer outage), the network queue will start back up and drain to the central archive computer with no loss of data.

Accounting for standard Ethernet bandwidth (~ 80 Gbytes/day), for the speed of writing to an Exabyte (~ 20 Gbytes/day) and for the overhead of the robustly conservative data handling practices designed into STB, the total throughput of either the KPNO or CTIO archive comfortably exceeds 10 Gbytes/day. By substituting a faster network, faster tape drive and/or a faster computer, this throughput could be increased significantly.

A monitor program, "bitmon", provides privileged access to the archive for swapping and verifying tapes and for performing various utility chores such as stopping the archive should the computer require servicing. STB is robust against tape drive failures, and can easily be reconfigured to produce different numbers of tape copies and to support additional tape drives (or pairs or multiples) to provide additional between-swap capacity. Only a single tape drive is actually required for normal archive operations. Duplicate copies are highly recommended, though, as the best way to safeguard an observatory's investment in its data.

In addition to the actual taped data files, STB produces two other data products—a FITS header catalog and a simple text index that cross-references the catalog with the particular tape ID, file number and image number within the large multi-extension FITS tape files. The header catalog format is easy to ingest into various commercial relational databases. The separate index allows for future recasting of the catalog without the painful need to read all the tapes, or for recasting the data tapes (perhaps onto DVD, for instance) without having to also recast a catalog database in a wholesale fashion.

The index file entries also serve as input to a simple IRAF CL script, "readbits", that prompts the user through mounting appropriate archive tapes as data are retrieved from the archive. Any image in the archive can be retrieved in a maximum of about ten minutes due to the efficient tape motions permitted by the large FITS tape files. Since STB produces standard FITS data files, any astronomical software package that supports FITS can be used to read the archive tapes and an IRAF installation is not required to operate the archive. The STB software itself requires only about a Mbyte of diskspace and could be installed on a quite modest workstation.

STB is easily portable to other operating systems (NOAO uses SunOS, and now Solaris, Sparcs) and to other astronomical instruments and telescopes, including those that may produce other than FITS data and potentially, other than imaging data. STB has also been successfully used with DAT tape drives, and would be easy to adapt to additional media formats. Compatibility with the Unix System V print spooler, **lpsched**, was provided for in the initial design and has been implemented by sites outside of NOAO. Our recent Solaris port of STB uses the excellent port of Berkeley **lpd** to Solaris that is available from:

ftp://ftp.eng.auburn.edu/pub/doug/

Current STB development efforts center on an update to support the WIYN Data Archive and Distribution System, which will use writable CD-R as the primary data distribution format for observers, as well as to construct an online random access data archive. We anticipate fielding this system at the WIYN telescope about the time this Newsletter is published.

This brief article has only touched on the many issues involved in operating an astronomical data archive 24 hours/day, 365 days/year. More information is available from *http://iraf.noao.edu/projects/stb*, or from the author at *rseaman@noao.edu*.

Rob Seaman

Layered Software Available for IRAF Versions 2.10 and 2.11

The following layered software packages are available from NOAO for installation in IRAF Versions 2.10 and 2.11 (layered software packages are optional IRAF packages not part of the standard distribution, which can be installed into an existing IRAF system). The list below includes only the layered packages developed at NOAO. Numerous additional packages (e.g., TABLES and STSDAS from STScI) are available from other sites.

New IRAF software often appears first in layered packages and is later incorporated directly into the main IRAF release. Hence some of these packages may already be included in the version of IRAF in use at your site. Contact us at *iraf@noao.edu* if you are unsure what version of the software your IRAF system includes (we will need to know the type of computer you are using and the IRAF version number).

All these layered packages are available via anonymous FTP from the IRAF network archive on *iraf.noao.edu* (140.252.1.1) in the directory *iraf/extern* and have "readme" files containing instructions for transfer and installation. Contact the IRAF hotline at *iraf@noao.edu* for further information.

ADC CD-ROM utility package - Two IRAF tasks that extract data from Volume I of the ADC CD-ROM text version. See the article in IRAF Newsletter Number 12 (July 1992).

color - A prototype IRAF color image display package that provides conversion of three bandpass IRAF images to a Sun 24-bit RGB rasterfile format, a 24-bit to 8-bit compression algorithm and Floyd-Steinberg dithering, and an RGB 8-bit pixel dithering algorithm. These tasks allow rendering of three color images for display on common 8-bit color workstations.

crutil - The **crutil** package contains tasks for removing cosmic rays from images. Some are for single images and some are for multiple exposures.

ctio - A package of miscellaneous IRAF tasks developed at CTIO. Contact *ctioiraf@noao.edu* for additional information.

digiphotx - The latest version of the **digiphot** package containing any bug fixes. The current version is installed in V2.11. See the readme file for details on recent revisions.

finder - A package for measuring celestial coordinates from a CCD or other digital images. A plate solution derived from the HST Guide Star Catalog sources in the object field can be transferred to a list of program objects specified by the user. A typical use for the package is to measure coordinates in

preparation for observing with a multi-object spectrograph. This package requires IRAF V2.11. See the accompanying article in this Newsletter.

fitsutil - A small package containing multi-extension FITS tools. This package requires IRAF V2.11.

focas - A suite of programs for automatic detection, photometry, matching, classification, and cataloging of astronomical digital images. The program automatically generates a catalog of objects and includes tools for subsequent analysis of the data in these catalogs. This is a collection of C programs that can be run either in the IRAF environment or outside it as a standalone package (Unix platforms only). Contact Frank Valdes (*fvaldes@noao.edu*) if you are interested in running FOCAS on a VMS platform.

imcnv - Two tasks for converting between various external data formats and IRAF images. These tasks, **import** and **export,** are included in the V2.11 release in the **dataio** package.

immatchx - A set of routines for doing image astrometry and matching image coordinate systems, point spread functions, and intensity scales using a variety of techniques. This package was included with IRAF V2.11 as part of the reorganized and extended IMAGES package.

mfilters - A set of routines for median/modal filtering of images using a variety of techniques. This package was included with IRAF V2.11 as part of the reorganized and extended **images** package.

mscred - The **mscred** external package is used to reduce CCD Mosaic data which is in the Mosaic data format. See the accompanying article in this Newsletter.

nmisc - Miscellaneous new tasks from NOAO collected together in a layered package to make them available prior to their release in the main IRAF distribution. Check the readme file for further information on what is provided.

rvx - The IRAF radial velocity analysis package. This is included in IRAF V2.10.3BETA and later versions of IRAF but is also available as a layered package for users with older versions of IRAF, or who want the latest version incorporating any bug fixes.

spptools - A collection of programming tools for IRAF developers working in SPP. Included are tasks to locate and print the calling sequences of procedures, reformat code in a standard format, create or query identifier databases, and create or rename external packages. Of special interest is the **spplint** task that can be used to perform a Lint-like compile time verification of SPP code. Contact Mike Fitzpatrick (*fitz@noao.edu*) for further information.

vol - A volume rendering package, used to visualize 3D data cubes. See the IRAF Newsletters Number 5 (October 1988) and Number 6 (February 1989) for further information. The tasks **im3dtran** and **imjoin** have been included in V2.11 as the tasks **im3dtran** (**imgeom** package) and **imjoin** (**imutil** package).

SAOimage - An X Window System based image display server for IRAF developed originally by SAO (see article in IRAF Newsletter Number 8, October 1989). This software is available in the *contrib* directory in the IRAF archives. (See also **Ximtool** from NOAO, and **SAOtng** from SAO).

uisdisp - Display software for non-DECwindows VMS Workstations. This software was discussed in IRAF Newsletter Number 7 (June 1989). It is included with VMS/IRAF Version 2.10. An update with some small bug fixes is available from the *contrib* directory in the IRAF archives. Contact Nigel Sharp (*nsharp@noao.edu*) for further information.

Additional layered packages from outside NOAO are also available in the *contrib* directory. The software in the *contrib* directory is contributed by persons outside the NOAO/IRAF group, and the author or authors of these packages are solely responsible for the software. Users with IRAF-related software they would like to share are encouraged to install their software in this directory (the directory is world writable). Even in cases where a software product is maintained and distributed elsewhere we try to include an entry in *contrib* pointing to the external software, to make it easier for people to learn about and locate such software.

The IRAF Group

IRAF NETWORK INFORMATION SERVICES	
IRAF Support:	mailto:iraf@noao.edu
IRAF Hotline:	(520) 318-8160
FAX:	(520) 318-8360
Network Archive (anonymous ftp):	ftp://iraf.noao.edu (140.252.1.1)
WWW URL:	http://iraf.noao.edu/
IRAFINFO Facility:	http://iraf.noao.edu/iraf-info.html
ADASS Newsgroups:	http://iraf.noao.edu/adass_news.html
Archive Listserver:	http://iraf.noao.edu/iraf-list.html
IRAF FAQ:	http://iraf.noao.edu/faq/

The *IRAF Newsletter* is published by the Central Computer Services, National Optical Astronomy Observatories, P. O. Box 26732, Tucson, AZ 85726. *Editors*: Doug Tody, Jeannette Barnes