AIPS++ is a project of the AIPS++ Consortium

Greenwich	GMT	+ 0:00
Australia	EAST	-10:00
Netherlands	MET	- 1:00
Charlottesville	EDT	+4:00
Urbana-Champaign	CDT	+5:00
Socorro	MDT	+6:00

AIPS++ Newsletter

Project Manager: Tim Cornwell
Managing Editor: Bob Hjellming
Production Editor: Kate Weatherall

June 1998

In this issue:

Status of the AIPS++ Parallelization Project

Doug Roberts - NCSA, Champaign

The AIPS++ parallelization group consisting of Dan Briggs, Doug Roberts, and Wes Young, overseen by Brian Glendenning, has been working on several components of parallelization. The effort to implement parts of AIPS++ on multiple processor machines had its first demonstration of a parallel application at the Alliance'98 Conference in Champaign, Illinois, on April 27-30. (http://alliance.ncsa.uiuc.edu/alliance98).

A poster with initial results and a future time-line was presented by Doug Roberts at the Alliance'98 meeting. The poster describes initial speed-up results on the SGI Origin2000 from a test parallel deconvolution (Clark `CLEAN`) on a VLA spectral-line data set. Parallelism in the `CLEAN` application was implemented using the Message Passing Interface (MPI). MPI is a portable system that allows data and instructions to be sent to remote processors (either on the same machine or on different machines). MPI is available on many different architectures ranging from a small network of workstations to large, massively-parallel machines. Because the MPI implementation of parallelism is portable, the same optimizations that are made for the large SGI Origin are useful on the relatively smaller machines of a typical astronomy group. The bottom line for the speed-up results is that the deconvolution of a large data set (127-channel HI absorption) scales well up to 32-processors (a speed-up of 22 on 32 processors - 70% of ideal). An on-line version of the poster is available at http://pacont.ncsa.uiuc.edu/A98/poster/roberts/.

In the short term (Summer 1998), our group is going to implement a parallel algorithm applicator class, using the MPI parallel `CLEAN` as a prototype. New classes derived from this parallel base class will carry out parallelization with a minimal effort from application programmers. This will address most "embarrassingly" parallel problems, notably spectral line processing. We are also investigating tuned libraries (starting with FFT's), notably the SGI/Cray Scientific Library (`SCSL`).

Our longer-term goals (Fall 1998) include parallelizing the `sky` application to carry out "embarrassingly" parallel deconvolution, and to do parallelized gridding and parallel FFT's. Also, we will be parallelizing a wide-field imaging algorithm (where the assumption that the sky can be represented at a single tangent plane breaks down). Finally, we will be increasing the user support to help astronomers, who need the large computational resources of NCSA using the parallel AIPS++ system on the Origin2000. This user support includes increasing the network bandwidth from the VLA to NCSA by putting NCSA on the NRAO intranet. By June, we should have installed a DLT tape drive on an NCSA machine, to which we have access. This DLT tape can be used to transfer larger data sets than can be stored on a single exabyte.

The Alliance '98 conference was attended by Dan Briggs, Dick Crutcher, Brian Glendenning, and Doug Roberts of the AIPS++ group, and Ruth Milner of NRAO. During the conference there were many discussions between the AIPS++ parallel group and other Alliance teams. Potential collaborations include a port of AIPS++ to Windows NT. NCSA has a new, large NT cluster that shows good speed-up using MPI on some software applications. A port of AIPS++ to NT would facilitate testing of AIPS++ on this NT cluster. This would be an important comparison of the high cost-performance of the NT cluster relative to the expensive, integrated distributed-shared memory architecture of the SGI Origin2000 system. Another connection made at the meeting was with the Parallel I/O group, headed by Dan Reed (UIUC CS department chair). This group has software that can be used to evaluate our code to identify I/O patterns and statistics. This would be important to show where performance due to increased I/O would be possible. The next generation of the MPI standard (MPI-2) includes a standard for MPI I/O. The MPI I/O standard is finalized and implementations are available. We intend to explore its use as a complement to the parallel processing development. NRAO will be hiring a programmer to work with Dan Reed's group to instrument the AIPS++ I/O libraries and investigate parallel I/O within AIPS++

Project News Summary

Tim Cornwell- NRAO, Socorro

Brian Glendenning is leaving the AIPS++ Project to join the NRAO Millimeter Array project as head of computing. This is sad news for AIPS++ but an excellent opportunity for Brian. He has been a member of the AIPS++ Project since its inception in early 1992, and has been our technical leader during all of that time. Since 1995 he has been Deputy Project Manager. Most of the architecture of AIPS++ originated in Brian's head and he has written much more than his fair share of the code. Beyond these direct contributions, he has been influential in all areas of the project by virtue of his willingness to comment on and offer advice on all aspects of the system. I'd also like to say on behalf of all of us working in the Project that he's been a complete pleasure to work with and we'll miss him in many ways. We wish him the best in his new work on the MMA. The computing needs of the MMA are daunting, both in hardware and software, but I can think of no one better suited to oversee the development of the computing systems.

Although Brian is a hard act to follow, I'm pleased to say that we have found a way to replace him. This requires splitting his two roles: Deputy Project Manager and technical leader:

- Athol Kemball will replace Brian as Deputy Project Manager, effective July 1. Athol joined the AIPS++ Project last year from Classic AIPS and has been directing our synthesis development effort. He has extensive experience in software development from his years working within Classic AIPS. In addition, he has a strong background in many technical aspects of interferometry and an active and productive scientific research program.

- Ger van Diepen of NFRA will become AIPS++ Technical Leader, effective September 1. Ger has been a key member of AIPS++ from the start of the project. He has contributed in many areas, in particular the AIPS++ Table system and more recently the Image classes.

In other project news, the third beta release is still in preparation. The highest priority items for the release are improved user documentation, more capable cross-calibration software, and a command-line interface to the object system. We are being helped in complete testing of the core parts of the system by a group of testers at the AOC in Socorro.

Measures, Part 2

Wim Brouw - ATNF, Epping

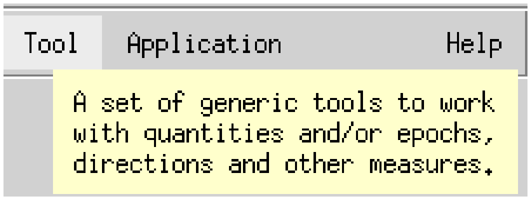
The functionality of the `Measures` module can be used interactively with the aid of a GUI. The GUI is started by typing `dm.gui()` on the ¡i¿Glish¡/i¿

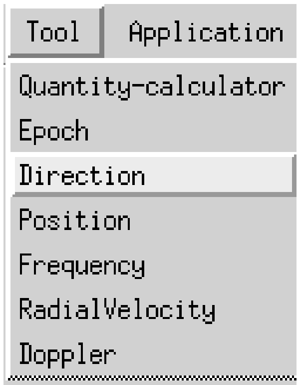
command line. It can also be started automatically by including in your *.aipsrc* the following line:

```
measures.default:  gui
```

The main menu bar below appears. When your cursor hovers above one of the menu items, a short description of the menu's contents appear. *(Due to rapidly changing nature of* AIPS++*, some of these GUIs may appear a bit differently.)*

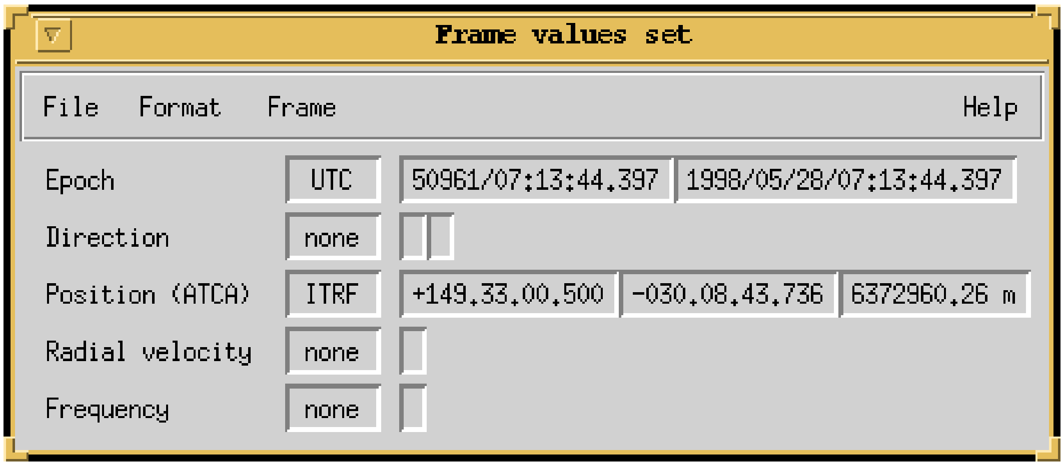
As an example, below on the left you see what happens when your cursor hovers above the *Tool* menu. If you press e.g. the *Tool* menu, a list of options appear as shown to the right:





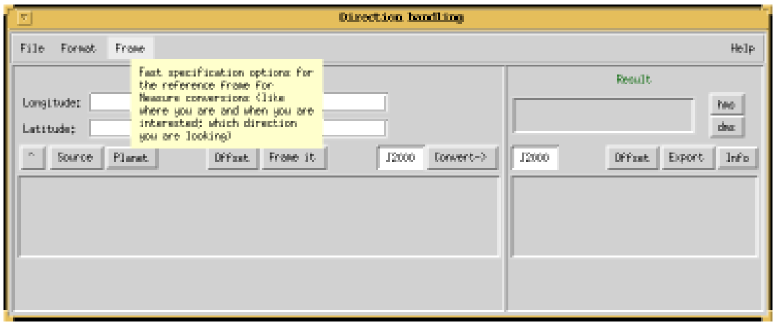
Going back to the main menu bar. The *File*, *Format*, *Frame* and *Help* items are on all windows. The *File* menu lets you close window(s). The *Help* menu offers you links to the reference manual (and options to stop the context help). The *Format* menu lets you select your favorite output format for results (check e.g. the frequency ouput options). Note that an ellipsis (...) indicates that you can fill in your own units.

The *Frame* menu lets you select your environment (location, etc.). Note that some actions automatically select 'now' as a frame option, and you can also select your location automatically by including your preferred Observatory name in your *.aipsrc* file in `measures.default.observatory`. Once you have actively selected a frame element, a frame panel appears (as shown at the above) , showing you the active frame elements.

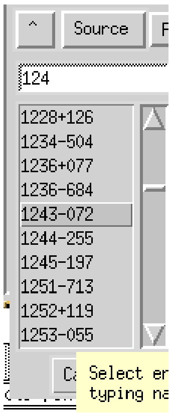


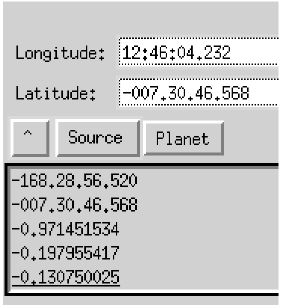
The *Application* menu gives a set of pre-programmed options like Clocks, Rise/Set times, and Velocity-to-Frequency conversions.

The *Tool* menu has options to open one or more specialized tools for a particular Measure (and also a Quantity calculator). Each of the Measure tools show the same format. They have the standard top menu bar as described above. In addition, they have an **input** section on the left, and an **output** section on the right. Results are obtained by pressing the `Convert-` button. *Direction Handling* GUI looks as follows:



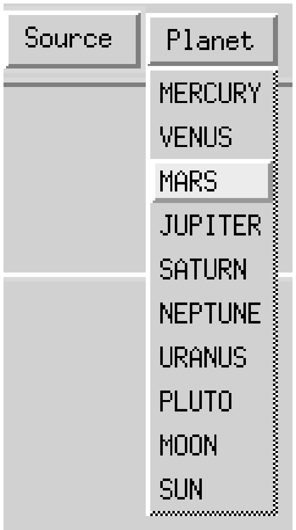
The white fields on the left are to input data (in the right units!). The field at top-right displays the result of a conversion in standard formats (either default or as selected by the *Format* main menu options). In general, the result fields have buttons to their right displaying the selected format or units. Pressing these buttons gives you the option to change them. The grey fields at the bottom of both sections display the input or result in alternative formats (see the hover help for the meaning of the fields).





The **Source** and **Planet** button in the left part of the GUI are special action buttons.

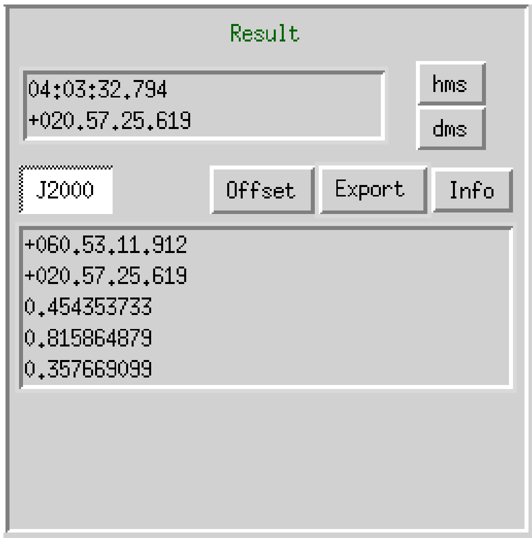
These buttons are selected for special short-cut inputs. In this case, you select a source from the source list (as shown above on the left). Selecting the source from the list fills in the coordinates (as shown above on the right).



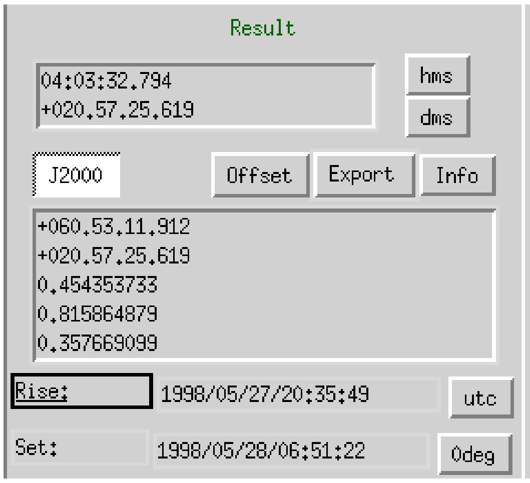
A planet can be selected with the `Planet` button (no input parameters appear because they are then unknown). You can type in your coordinates manually by selecting the white button (shown as J2000, by default) and choosing the reference code of the input. When loading sources or planets this field is automatically loaded. The `Offset` button selects the input as an offset to be added to subsequent inputs (especially useful for Epochs). The `Frame it` button uses the input as an entry in the reference frame.

If you provide the values in wrong units a pop-up appears.

When you select a planet, a `Frame` window appears giving the position of a planet. You must specify the time. If you do not select an epoch, the GUI assumes you want the time as 'now'.



As an example select an input, say the planet Mars. To get the position of the planet, you have to convert it, first by selecting an output reference code in the right-hand white selector field, and then pressing the `Convert` button. Output values as shown on the above, appear. The small `hms` and `dms` buttons can be used to control the output format. The middle buttons act on the result. With the white button you can select another reference code (and press `Convert` again). The `Offset` button lets you specify an offset (taken from the input field!) which is subtracted from the result before it is shown.



The `Export` button, shown below, exports the result to a (selectable) *Glish* variable that can be used on the command line. The `Info` button lets you select additional information (e.g., Rise, Set, Co-latitude, and User). On the left is an example if you select the Rise (or Set, they are coupled) information. Note again the small buttons on the right, which in this case allows you to specify the type of time you want, and the elevation limit. All Info buttons have a `User...` button, which lets you define a *Glish* function that will be called (the name is selectable, as is the unit option). For details of the interface between the function and the result see the Reference Manual.

This report tries to give some flavour of the Measures GUI (and its associated code). I hope it helps you in pursuing its possibilities further. Please let me know of any extension you would like to see (or bugs you find).

Newsletter Mailing List

Tim Cornwell - NRAO, Socorro

To alert our readers to each new issue of the AIPS++ Newsletter, we have inaugurated a newsletter mailing list. When a new edition is published, we will send an announcement to this mailing list. To subscribe to the mailing list, send mail to **majordomo@majordomo.cv.nrao.edu** with the following command in the body of your email message:

`subscribe aips2news`

If you ever want to remove yourself from this mailing list, you can send mail to **majordomo@majordomo.cv.nrao.edu** with the following command in the body of your email message:

`unsubscribe aips2news`

If you ever need to get in contact with the owner of the list, have trouble unsubscribing, or questions about the list itself, send email to **owner-aips2news@majordomo.cv.nrao.edu**.

The Story of *Glish* - Part 3

Darrell Schiebel - NRAO, CV

The two previous installments covered the origin of *Glish* and the concept of *Glish* as a **glue** language. In this installment, I would like to discuss the latest addition to *Glish*'s functionality, and in the process, try to illuminate *how Glish* evolves.

Early in the AIPS++ project, we had difficulty with GUIs (graphical user interfaces). It was difficult to choose which base to build our system on, e.g. *Motif* or *InterViews*, GUI builder or library. It was about this time that I added the `Tk` widgets to *Glish* because it could become the basis for **simple** GUIs inside AIPS++ and because I thought anyone using *Glish* should have access to the GUI elements. The Tk widgets were the obvious choice because they were widely used and because they were built for the sort of interactive use I planned.

The mapping of a GUI onto *Glish*'s event mechanism was obvious and painless. Indeed, most GUIs are written in an event-oriented style where functions are registered for all of the important GUI elements. These `callback` functions are then called as the user interacts with the GUI. This is exactly what happens with *Glish*'s `whenever` statements. For example,

- `f := frame()`
- `b := button(f,'press me')`
- `whenever b->press do`
- `print 'Hello world!'`

this script displays a button, and each time the user pushes the button "Hello world!" is printed. Each of the Tk widgets becomes a *Glish* client which sends and receives *Glish* events. This required several changes to *Glish* because, up to this point, all clients were separate (operating system) processes which were connected to *Glish* through inter-process communication (IPC) connections. Another thing to note about my plan for the interface is that one widget is placed inside of another by passing the outer widget to the inner widget as a parameter. The outer widget is, in a sense, the **parent** of the inner widget. In this case, the frame, `f`, is passed as a parameter when the button, `b`, is created. This looks like:

First the top level frame is created, and then the button is placed inside of it. This means further changes to *Glish* because it requires that clients be able to be **passed** to other clients.

None of this functionality, i.e. clients of smaller granularity than a process, and passing clients to other clients, was available in *Glish*. The proper design for this would be to build a client that would act as a repository for these GUI widgets. When the user creates a widget agent in a script, the actual Tk widget would be created in the client. Events sent to any of the GUI agents created in the script would be forwarded to a single client process. Likewise any events, generated by user interaction with the GUI, created in the client would be forwarded to the proper agent in the script. The advantages of this approach would be that the X libraries would only be needed for the GUI client so the *Glish* interpreter would remain small. Another advantage would be that the flexibility of clients would be greatly improved because any client could act as a repository of agents, each of which would be a conduit of events to the repository process.

However, it wasn't at all clear if the GUI clients would be acceptable to programmers, or if the resulting system would be fast enough for interactive use. Also since much time had been wasted with *InterViews* and *Motif,* this widget binding needed to be fashioned quickly. As a result of these pressures and a general lack of through experience with *Glish*, the easiest way out was taken. The `Tk` widgets (along with the `X` libraries) were linked into the interpreter. This nearly doubled the size of the interpreter, but the widgets were available for *Glish* users in about three weeks. Since that time the `Tk` widgets have provided the basis for most AIPS++ GUIs.

In general, the use of these widgets has been very successful. GUIs have been created by users who probably would not have attempted to create a GUI using *Motif*. The `Tk` widgets have been very robust. There were some problems, and it is these problems that provided the impetus for fixing things properly by moving the widgets to a client.

The *Rivet* library (`http://personal.veritas.com/ brianw/rivet/`) was chosen for the initial binding of the widgets to *Glish* because it throws out the `TCL` interpreter while keeping the `Tk` widgets. It was hoped that this would be faster while avoiding linking in the `TCL` interpreter library. It is doubtful that the *Rivet* widgets would have been any faster, and the savings in executable size was probably minimal. The final nail in *Rivet*'s coffin came when the author quit supporting it, and it fell far behind the release of `TCL/Tk`.

Some GUIs proved to be too slow with the *Glish* based `Tk` widgets. The best example is the table browser that reads data from disk and uses it to fill in a spreadsheet-like GUI. The user can then interact with the data through the spreadsheet. This GUI was too slow mainly because of the all of the operations necessary to initialize the interface.

Another problem with the `TCL`-less *Rivet* based widgets was that *Glish* users couldn't easily get access to all of the `TCL/Tk` widgets which are readily available. Some of the **mega-widgets** that are available could simplify some complex GUIs, like the table browser, and make them much speedier. These **mega-widgets** are packages of multiple smaller widgets, and they implement more complex GUI components, like a spreadsheet. These **mega-widgets** are implemented using `TCL`, and sometimes compiled C code.

Moving these widgets to a client and moving from *Rivet* to `TCL/Tk` directly solves many of these problems: an unsupported product is exchanged for a supported one; **mega-widgets** implemented in `TCL` and **C** may make large GUIs fast enough; *Glish* users can get access to all of the `TCL/Tk` widgets easily; the *Glish* interpreter's size (since it no longer needs to link against `X` and `TCL/Tk` libraries) is cut in half; and *Glish*'s client interface is expanded and improved.

The first steps of this conversion are finished. A *Rivet*-based client application has been created which serves up the widgets to the interpreter. This setup seems fast enough, but the table browser is slower. A `TCL/Tk`-based client application is being created and should be completed shortly. In the near future, this change will be integrated into AIPS++ after there has been sufficient testing and debugging.

AIPS++ Cookbooks Made of Recipes

Bob Hjellming - NRAO, Socorro

We are developing a prototype Cookbook for AIPS++ users based upon the concept of recipes that achieve particular goals. The following is an example of a recipe using a particular format. With this format every recipe begins with a statement of goals, a list of what is used from AIPS++, the results, and (most importantly) the assumptions (=ingredients) you must have. These recipes are intended to be readable in hard copy form, but optimum use is obtained when reading with a Web browser because of the links to data files or other documentation.

Recipe:

Goals: Read an ASCII data table into AIPS++ and make plots of the data

Using: *tablefromascii* function *Table*, and *PGPlotter*

Results: Data Columns in *Glish arrays, and plots of these data*

Assume: You have an input file of ASCII data in columns, and a header description file. In this recipe the ASCII data file named `1909+048 has seven columns of numbers. The ASCII header file describes the data columns in the data file; in this case a header file gbihdr assigns names and data types to each column, following the rules described in documentation of` *`tablefromascii`* `in the` AIPS++ Reference Manual header file contents. Both input and header files must be in the directory from which you started AIPS++

.

Script: A *Glish script executes this recipe.*

AIPS++/Glish commands and results	***Purpose and Background***
AIPS++ *Glish* `commands and results`	**Purpose and Background**

`tablefromascii('ss433table', '1909+048', 'gbihdr')`	Put ASCII tables of data into an AIPS++ table is using the AIPS++ `tablefromascii` function. For details on what you can do with AIPS++ tables see the *Module Reference Manual* at http://aips2.nrao.edu/aips++/docs/user/Refman/. Invoking `tablefromascii` results in an AIPS++ Table named `ss433table` written to disk.
`ss433 := table('ss433table')`	Read this Table into a *Glish* object named `ss433`
`ss433.getkeywords()`	Show keywords defined in the gbihdr file

[INSTRUMENT=Green Bank Interferometer, FREQUNITS=GHz, FREQS=2.25, FREQX=8.3, FLUX=average of RR and LL, MJD=average Julian Day - 2400000.5 for scan, SS=2.25 GHz FLUX, SX = 8.3 GHz Flux, LHA=Local Hour Angle for Scan]

Resulting listing of keywords
`mjd := ss433.getcol('MJD')`
`Ss := ss433.getcol('SS')`
`Sx := ss433.getcol('SX')`
	Extract columns from the table and assign them to 1-D arrays using *getcol* on the *Glish* table object named `ss433`

`plot := pgplotter(background='white')`	Make a pgplotter object named plot and open a plotting window on the screen

`xmin := min(mjd)`
`xmax := max(mjd)`
`ymin := 0`
`ymax := max(Ss)*1.05`
	Calculate range variables for x and y axes

`plot.env(xmin,xmax,ymin,ymax,0,1)`	Set plotting environment, mainly to change the range for x- and y- axes
`red := 2; blue := 4`	Set color values
`plot.sci(red)` `plot.line(mjd,Ss)`	Set plotting color to red and make a line plot of Ss vs mjd in the plotting window
`plot.sci(blue)` `plot.line(mjd,Sx)`	Set plotting color to blue and add a line plot of Sx vs mjd

In the Pgplotter window, *File* can be used
to print, save, etc., the plot(s).

The development of recipes has been slow for interesting astronomical applications because of the changes and improvements in major modules like `ms`, `cal`, `sky`. A set of astronomical recipes for doing standard things should be available sometime during the Summer of 1998. Contributions of recipes or other material for this prototype AIPS++ Cookbook are welcome. Please send them to *rhjellmi@nrao.edu*.

What's New in AIPS++

Tim Cornwell - NRAO, Socorro

The following personnel changes have occurred:

- Brian Glendenning is leaving AIPS++ to join the NRAO MMA Project.
- Athol Kemball (NRAO) will become Deputy Project Manager, effective July 1.
- Ger van Diepen (NFRA) will become AIPS++ Technical Leader, effective September 1.
- Peter Barnes and David Barnes (no relation!) have joined AIPS++ in Socorro and Epping respectively.
- Friso Olnon (NFRA) has left AIPS++ to work at the Joint Institute for VLBI in Europe.

The following changes have been made to the AIPS++ development system:

- `Sky` has an "advise" function which can suggest imaging parameters (e.g., `cellsize`) for you. The speed of `sky` has been improved in several areas, and some subtle bugs have been fixed.
- We have improved our "component list" implementation. At present a component list may consist of an arbitrary collection of point components and Gaussians, possibly with a spectral variation. `Sky` can use a component list as a model. AIPS++ will probably distribute calibrator lists in this format.
- The `Dish` program can now save and restore its state.
- The `Measures` GUI can now handle source lists. The `Measures` system now understands the ITRF reference frame. We have also added a `Measure` for the Earth's magnetic field, and a number of `Measure` classes to aid calculation of baselines and `uvw` vectors.

- A simple `plotxy()` function has been added to `pgplotter` to make it simpler to plot vectors from *Glish* (e.g., you no longer need to call `pg.env()`). `pgplotter` should now almost always be able to use a shared color map (no flashing) as long as you don't need too many colors (in particular, if the number of colors is 16 or less).

- *AipsView* now works with the `CDE` window manager, and has better color map control in general.

- A rich set of `Image` mask and region classes have been checked in. Although there is no visible change to users yet, these classes allow construction of sophisticated image analysis tasks.

- Progress bars (used for, e.g., gridding) now only appear after 5 seconds, to prevent visual clutter if you are doing many short operations.

- ASCII File I/O and formatting (`printf` style) has been added to *Glish*. *Glish* regular expression matching has been improved (e.g. so the result can now be used as a mask). For *Glish* programmers, $value, $agent, and $name are now local to the `whenever` scope, not global, so you do not need to worry about them being clobbered by intervening events or function calls.

- You can now change values in `Tables` from the new table browser (`newtb`). You can also export and import `Table` columns to a *Glish* variable from the browser.

- The "no frames" versions of the AIPS++ web pages are greatly improved. Let us know what you think! They will become the default pages in the near future.

- The project has been experimenting with the `egcs` compiler (`GNU` with additions from *Cygnus* and other sources) with good results. In particular, native exception handling works on at least some architectures, which improves the reliability of the system. We are considering releasing `egcs` binaries for the next beta release.