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

Greenwich	GMT	+0:00
Epping	EAST	-10:00
Netherlands	MET	-1:00
Charlottesville	EST	+5:00
Champaign	CST	+6:00
Socorro	MST	+7:00

AIPS++ Newsletter

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

November 1998

In this issue:

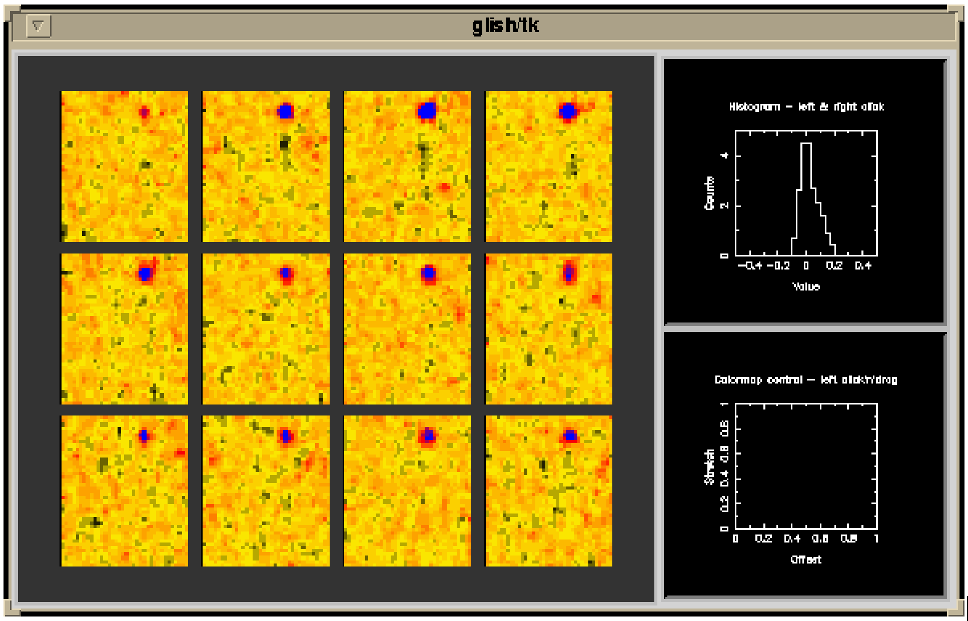
The AIPS++ Display Library - Part 1

David Barnes - ATNF, Epping

The AIPS++ Display Library was introduced in the March 1998 AIPS++ Newsletter. Since then, a substantial amount of design and implementation work has been devoted to the Display Library. The purpose of the Display Library is twofold:

- to provide a **C++** library of high-level classes for the display of astronomical datasets, with which **C++** programmers can build stand-alone visualization applications, and

- to provide an *interface* to this library, which brings the bulk of the functionality of the **C++** library to the *Glish* command line, and with which *Glish* programmers can build visualization applications that are wholly integrated into the AIPS++ data reduction and analysis environment.



[H]

The purpose of this article is to outline the current status and future design for the *Glish* interface to the AIPS++ Display Library. A future article will explore the **C++** interface to the Display Library.

The *Glish* interface to the Display Library is in its infancy: since April, development has been in full swing and has culminated in a "first generation" *Glish* widget which is capable of displaying different views of multi-dimensional AIPS++ Images and in-core *Glish* arrays. Mouse controlled zooming and *Glish* controlled animation have been demonstrated, together with the capability to use various built-in and user-defined colormaps. Presently, the *Glish* interface to the Display Library is undergoing a proper design and specification, and the resultant "second generation" widget should be available, with substantial functionality, in the AIPS++ system in or shortly after November 1998. Briefly, the implementation of the interface as a dynamically loadable *Glish* widget means that Display Library canvases can be embedded directly in graphical user interfaces developed in *Glish* code, and are programmable in the normal *Glish* event-oriented fashion. As development moves into the applications stage, general and specific *Glish* wrapper scripts will be generated which can be used to hide the direct widget interface from the user. In this article, code fragments are given which can be used to generate an interactive "cgdisp"-like panelled view of a multi-dimensional dataset, as shown in the figure above.

This script commences by building a frame containing one pixelcanvas widget and two **PGPLOT** widgets:

```
f := frame(side='left');
pc := pixelcanvas(f, [width=400, height=340, background='gray20']);
```

```
fr := frame(f, side='top');
pg_hist := pgplot(fr, width=180, height=170);
pg_shap := pgplot(fr, width=180, height=170);
```

A canvas for drawing astronomical data can be added to the pixelcanvas widget as follows:

```
wc[1] := worldcanvas(pc,0.2,0.2,0.5,0.5);
```

where the first parameter indicates that the worldcanvas is to be added to the existing pixelcanvas `p`, and the remaining four parameters comprise the (fractional) bottom left and top right corners of the canvas on the visual widget. Multiple canvases can be added to a single widget; it is left as an exercise for the reader to write code to set up the panels as shown above. Let us assume then that the twelve panels in the figure above are stored in the array `wc[1:12]`.

The next task is to create what is known as a "DisplayData" object, which will be a *Glish* handle (technically a *proxy*) to an instance of a **C++** class within the AIPS++ Display Library. We take an R.A./Dec/Frequency AIPS++ Image, `/data/testdata.im`, and construct a DisplayData which will draw raster images of the Frequency planes of the dataset.

```
mydd := displaydata('raster', '/data/testdata.im');
```

and add this data to the 12 panels:

```
for (i in 1:12) {
    wc[i]->add(mydd);
}
```

At this point, the default colormap, `heat`, has been used, and there are 12 identical slices of the Image displayed on the screen. You can modify each canvas so that it displays a different plane of the dataset, and use a different colormap, e.g., `smooth`, using the following code:

```
for (i in 1:12) {
    wc[i]->options([element=i]);
}

mycm := colormap('smooth');
mydd->setcolormap(mycm);
```

You can now make use of the AIPS++ Images module to calculate a histogram of the pixel data values in the Image, and then display this histogram to using the first of our **PGPLOT** widgets. At the same time, you "bind" to left and right mouse button click events on the **PGPLOT** widget to set, respectively, the minimum and maximum data values for display:

```
include 'image.g';
im := image('/data/testdata.im');
abslimit := 0.5;
im.histograms(histout=myhist, nbins=30, includepix=abslimit,
              async=F);
myhist.counts := log(myhist.counts + 1);
t := pg_hist->env(-abslimit, abslimit, 0,
              as_integer(max(myhist.counts) + 1), 0, 0);
t := pg_hist->lab('Value', 'Counts',
              'Histogram - left &amp; right click');
t := pg_hist->bin(myhist.values, myhist.counts, T);
t := pg_hist->bind('&lt;Button-1>', 'left');
whenever pg_hist->left do {
              left := $value.world[1];
              mydd->options([datarange=[left,right]]);
}
t := pg_hist->bind('&lt;Button-3>', 'right');
whenever pg_hist->right do {
              right := $value.world[1];
              mydd->options([datarange=[left,right]]);
}

# set the initial data range:
left := -0.02;
right := 0.02;
mydd->options([datarange=[left,right]]);
```

Likewise, you can use the second PGPLOT widget to control the shaping of the colormap itself, as opposed to the mapping of the data values onto the colormap. Using pg_shap, arrange to bind to left mouse button click and drag events, and calculate new parameters for a linear colormap ramp function:

```
function shape_colormap(world) {
     world[2] := tan((world[2] - 0.5) * pi);
     if (world[2] == 0) {
         world[1] := 0.5;
} else {
     world[1] := world[1] + (world[1] - 0.5) / world[2];
}
mycm->options([shapecoeffs=world]);
}
```

This concludes the example. Very little code remains for the user to generate in order to reach the working example shown in the figure. I hope this article has demonstrated the relative ease with which users and programmers alike are able to rapidly generate simple or sophisticated visualization applications using only *Glish* scripts.

Present and near future work on the Display Library will be concentrating on the full specification of the visual widget, and its helper agents - canvas, colormap and lattice. Major tasks prior to the widget being made available in the AIPS++ distribution include implementation of these specifications, together with extensions to the Display Library infrastructure to support contour and vector field drawing, multiple dataset overlays, axis labelling, and ImageRegion display.

Summary of the Third Beta Release of AIPS++

Tim Cornwell - NRAO, Socorro

For newcomers, this is an overview of the contents of the third beta release of AIPS++. Included in this release are:

- A powerful command line interpreter called *Glish*, the heart of AIPS++, which has both a tutorial and reference manual.
- Objects allowing users to to do things such as start, control, and stop objects is via the Object Catalog (`objectcatalog`), whereas files are handled by the File Catalog (`catalog`).
- Tables are how all AIPS++ data are stored and made accessible from inside *Glish* via the Table `object`.
- Graphical User Interfaces (GUIs) for most user-visible objects, apart from the *Glish* command line interface. These represent a large investment of effort in direct response to our earlier beta releases where we learned that users needed simpler interfaces.
- Synthesis imaging capabilities are found in the two `objects` Imager and Calibrater, the principal objects for imaging and calibration. Imager, in particular, has many features not found in other packages, such as multi-field processing in user-selected coordinate systems. We expect Calibrater to evolve similarly over the next few months.
- The Measures System is, to our knowledge, unparalleled in the types of operations it allows on measured quantities with units and coordinate systems. The functionality is available via both the *Glish* command line (the Measures `object`), and a customized graphical user interface.
- The single dish environment `dish` is a carefully designed interactive tool for processing of single dish data, most particularly spectra but other one-dimensional data formats are supported.
- A very capable plotting tool, Pgplotter, based on the **PGPLOT** subroutine library, is available and is used by many other tools. For example, an interactive visibility plotting tool, visplot, has been built on top of Pgplotter.

- Numerous general purpose utilities for handling AIPS++ data are present. For example, the tablebrowser (available as a function **browse** of the `table` object) can edit, plot, query and select data, and can be configured in many ways to display data as the user desires.

- To demonstrate the possibilities of stream-lined processing in AIPS++, we have provided the **simpleimage** `object`. This walks the user through straightforward imaging of synthesis data, first making a tapered image at low resolution and then imaging a user-specified region at full resolution. The user has only to select the data to be imaged, and to delimit the full-resolution region. This is an example of the type of packaging that we plan to incorporate in the future.

- Introductory documentation has been written by early testers of AIPS++, and terminology established for subsequent use throughout the project.

- Extensive reference documentation is available for all user-visible objects.

- The system has powerful programmability at the *Glish* level.

Not (yet) included in this beta release are:

- Filling of VLA or VLBA data apart from via UVFITS. Fillers for other telescopes and formats (i.e., **ATNF, BIMA, WSRT, UVFITS**) are present. Currently we expect you to apply AIPS++ to data that has been initially calibrated elsewhere.

- Efficient processing of large synthesis continuum images (e.g., *millions of pixels*) or many channel (e.g., *more than 64*) spectral images. Current processing is optimized towards spectral-line observations of moderate numbers of channels.

- Sophisticated cross-calibration capabilities.

- Interactive flagging of visibility data.

- A robust way to archive the MeasurementSets to FITS. The best is to use UVFITS for synthesis data (`ms.tofits`) and `ms2sdfits` for single dish data.

- We do not expect people to program at the **C++** level using the beta release.

Getting Started in AIPS++ Documentation

K.R. Anantharamaiah - Raman Research Institute, Bangalore and NRAO, Socorro

A new version of the document *Getting Started in* AIPS++, intended for newcomers to AIPS++, has been released and can be found in the AIPS++

documentation system. This document provides an introduction to AIPS++, practical information about how to start AIPS++ on your workstation, and how to use AIPS++ for data processing applications. The reader is taken through a series of simple steps consisting of:

- starting AIPS++ and useful initial steps;
- creating a simple script;
- generating a test uv-data set;
- making an image from the test data;
- examining the uv-data set and the image;
- creating a simple plot;
- creating an AIPS++ object using some Object functions;
- using the *Glish* command line and using GUIs; and
- doing some simple computations using *Glish* commands.

The document also introduces the reader to both some basic concepts and the structure of AIPS++, which are slightly different from other data processing packages. It is the first document that should be used by astronomers trying the new beta release.

Getting Started in AIPS++ is written in a semi-tutorial style and thus, for maximum benefit, should be read during a "live" session with AIPS++. At the end of such a session, the reader is likely to have gained some feeling for how AIPS++ works and be ready for more advanced use of the package.

Project Summary

Tim Cornwell - NRAO, Socorro

The AIPS++ Third Beta was released on October 15, 1998. This is the culmination of a year long period of development, for the most part along lines we had anticipated in our planning, and also partly in response to comments from testers of our previous beta releases. This release sees substantial improvements in all areas, most importantly in interfaces and in functionality. Details of the contents of the release are given in the above newsletter article in this issue. Our beta testers are volunteers from both inside and outside the project. If you would like to participate in the beta test program, please contact Tim Cornwell at tcornwel@nrao.edu.

After this beta release, we will switch to a new approach to release. Instead of a fully-fledged beta release, we will instead issue binaries for the latest stable build. This shift reflects the stabilization of the AIPS++ infrastructure as well as an increased emphasis on applications development and attracting new users. We hope to be able to make stable releases at a rate of about one per month to 6 weeks. Our expectation is that this will enable beta-testers to use the system more consistently, as both new features and bug fixes will be made available more expeditiously. All of the testing and continuing development is aimed at a public release of AIPS++ by March 15, 1999.

We're pleased to have selected a logo for the Project: see the top of this newsletter. You will start seeing this logo throughout our documentation.

Finally, we plan demonstrations at the ADASS meeting in Champaign-Urbana, early November, and at the AAS meeting in Austin in January. If you are attending either one of these conferences, please come on by and see us.

The AIPS++ Table System

Ger van Diepen - NFRA, Dwingeloo

In the beginning of the AIPS++ project it was decided that all persistent data should be stored in a common form since it would make it possible to have a uniform interface to all data and to develop common tools to handle data. Allen Farris (from StSci) developed some of the initial ideas about a table system; it was based on relational database concepts, but extended those concepts in various ways. Most of Allen's ideas were used and implemented as the AIPS++ Table System.

Some examples of tables now being used in the AIPS++ system are:

- The measurement set is a table with a collection of sub-tables;
- An image is a table; and
- The log messages are saved in a table.

Furthermore the Westerbork Telescope Management System (http://www.nfra.nl/tms) has adopted the AIPS++ Table System as the means to store data. This article will only give an introduction to the table system. It is beyond its scope to explain the possibilities of the Table System in all its detail. However, in the next issue(s) of the Newsletter, some examples will be given how to use it from **C++** or *Glish*. This article contains several links to AIPS++ documentation explaining things in more detail.

Structure of a Table

An AIPS++ table has the following structure:

1. A set of so-called table keywords contains global information about the data. They can be compared to the **FITS** headers, but are not limited to **ASCII** scalar values.

2. A set of columns and rows contains the actual data. They can be compared with the **FITS** binary tables, but offer more flexibility.

3. Each column has its own set of so-called column keywords containing information about the column (e.g. unit). Column keywords are used by the TableMeasures module to handle measures in tables. A keyword set is a TableRecord `object`. It is a heterogeneous collection of names and values. Each value can be a scalar or N-dimensional array of one of the basic data types (`char, bool, short, int, float, double, complex, double complex, and string`). The value can also be a TableRecord in itself making it possible to define hierarchical keyword sets. Finally a value can be a reference to another table making it possible to have a hierarchy of tables. The latter is, for instance, used heavily in the Measurement Sets.

4. Columns have the following characteristics: They can contain scalar or array values. An array can be of any shape and dimensionality. It is possible to force all cells in a column to have the same shape or dimensionality. All basic keyword data types can be used for a column. Before the end of 1998 it will also be possible to use a TableRecord as the data type of a column to make it possible to store heterogeneous data in a column.

Storing a Table

The Table System makes a clear distinction between the logical and physical layout of the data. The logical layout is defined by creating a table description. The physical layout can be defined at table creation time by binding columns to so-called storage managers. The storage managers control how the data are stored. By choosing the correct storage manager, you can optimize in access performance and/or disk space. The following storage managers are available:

1. The simple storage manager StManAipsIO. It writes each column as a series of values into the disk file. Currently this storage manager is memory based, but it is planned to make it truly disk based.

2. The disk space saving storage manager IncrementalStMan. It only writes a cell in a column when its value is different from the value in the previous cell. It can save quite some disk space when the values in a column are constant or vary slowly.

3. The array based storage manager TiledStMan. This storage manager is optimized to store large arrays which must be accessed efficiently along various axes. It is, for instance, used to store images to make access in RA, DEC, and frequency almost equally fast. It is also used to store the data in a Measurement Set.

Apart from binding to a storage manager, a column can also be bound to a so-called virtual column engine. Such an engine makes it possible to calculate the data on-the-fly. An example of it is to store a float column as 2-byte integers by scaling the values.

Other functionality provided for Tables

It is possible to share a table amongst multiple processes. The system maintains a read or write lock on the table to synchronize access to the table when multiple readers and writers are active at the same time. Currently synchronization is fully supported for `get` and `put` operations, but not yet supported for cases where the structure of the table is changed by adding or removing a column. That kind of synchronization will be added soon.

A table can be sorted on any combination of scalar columns. It is also possible to select a subset from a table using an SQL-like selection mechanism. Sorting and selection result in a so-called reference table. This reference table contains the row indices of the original table and can be handled as any other table. The user will not notice any difference compared to a 'normal' table.

A special kind of sorting/selection is offered by the table **iterator** making it possible to iterate through a table based on the contents of one or more scalar columns. Each step in the iteration gives a reference table containing the rows with the values in the iteration's columns.

How to use the Table System

Tables can be created and accessed at the **C++** and *Glish* level. The **C++** interface offers most flexibility, but the functionality offered in *Glish* is quite extensive. The selection/sort interface is called Table Query Languange (TaQL). It is described in AIPS++ note 199.

Synthesis I: The Measurement Equation in AIPS++

Athol Kemball - NRAO, Socorro

This article is the first in a series to consider issues of relevance to aperture synthesis processing within AIPS++, and is concerned with the overall role of the Measurement Equation in AIPS++.

All data reduction packages for radio-interferometric instruments implicitly adopt a mathematical model for the instrumental effects that are to be removed in the course of data reduction. In essence, they are algebraic relations describing how the stored calibration information is to be applied to the uncorrected uv-data to derive calibrated uv-data suitable for imaging or further analysis. The adopted calibration relations serves a secondary purpose also, namely in providing an assumed model for the propagation of instrumental errors affecting the observed data. In this sense, it is also reflecting the implicit assumptions being made regarding the physics of the observing process or instrument. This model is used directly, or implicitly, by the solvers that determine the calibration quantities from the observed data.

In practice, data reduction systems adopt calibration models that are sufficient for the instrumental systems under consideration. Instrumental effects are aggregated or approximated where possible, and effects below a certain threshold of significance may not be modeled. In addition to deciding which instrumental effects are to be included (e.g., instrumental polarization), a parametrization of each effect also needs to be adopted. This includes the coordinate dependence of the effect, such as time and frequency, and the model parameters characterizing the instrumental correction itself.

Important tradeoffs are involved in deciding the complexity of the calibration model to be adopted, both in terms of which effects to include, and their parameterization. Increasing generality allows a broader range of instruments to be covered, and allows greater flexibility in exploring more complete calibration models as any individual instrument evolves in sensitivity and flexibility. As an example, the VLA is operating at lower frequencies than previously supported, including 74 MHz. This introduces new calibration requirements beyond those required at centimeter wavelengths. Overly complex calibration models are, however, difficult to support in software, both in correcting data and as used by the calibration solvers. In addition, they are difficult for the end-user to understand, and so increase the difficulty of using the software.

AIPS++ requires a calibration formalism that is sufficient to support data reduction for a generic interferometer or single-dish. The framework for such a model has been laid in a series of papers by Hamaker, Bregman and Sault (Hamaker, Bregman and Sault 1996; Sault, Hamaker and Bregman 1996; Hamaker and Bregman 1996; and Hamaker 1998). This model provides a mathematical formalism for describing radio-interferometric polarimetry that offers several advantages over previous considerations of the problem. The formalism traces the signal path for each antenna and represents the instrumental effects as complex (2x2) Jones matrices. The Jones matrices have independent representation in an arbitrarily polarized basis (e.g. linear or circular), and can similarly have multiple parametrizations. The impact of the instrumental Jones matrices on the interferometer output for a given baseline is represented by the outer product of the individual Jones matrices to produce a (4x4) correction matrix. The Jones matrices act on 4-dimensional coherency vectors representing the polarization properties of each signal. A detailed description of each calibration component will be provided in the next article in this series, including the mathematical details involved. This formalism does not necessarily represent new physics, but does provide a general formulation that is independent of specific representations. This is a powerful advantage.

The Hamaker, Bregman, Sault formalism, as adopted by AIPS++, is referred to as the Measurement Equation. Central features of this approach include the use of Jones matrices to model instrumental effects, support for arbitrary Jones matrix parameterizations and a sufficiently general model to cover a generic interferometer. This instrumental model is used applying calibration corrections and by basic calibration solvers. The model includes both uv-plane and image-plane effects. This powerful generality has been carefully balanced against implementation questions raised in the discussion above. The objective in using this system is not and end in itself, but rather to provide a robust framework for implementing a powerful, and general, formalism sufficient for the AIPS++ project's objectives. Our experience in using this model in practice has borne out this expectation, and it has proved to be a practical solution to the question of how to support calibration of diverse radio-interferometers, with adequate room for growth in both the specialization of the calibration effects considered and their representation. This formalism allows new science to be explored, both through the inclusion of new calibration effects and parametrizations, but also the calibration of arrays with mixed instrumental effects such as different polarization types. The formalism can also be used for deriving and applying calibration to single-dish telescopes, and work in this area is underway.

The next article will consider the details of how the Measurement Equation is used in AIPS++, and the identification of the individual Jones matrices with commonly understood calibration effects.

References

Hamaker, J.P., Bregman, J.D., and Sault, R.J., 1996, *AAS*, **117**, 137.
Sault, R.J., Hamaker, J.P., and Bregman, J.D., 1996, *AAS*, **117**, 149.
Hamaker, J.P., and Bregman, J.D., 1996, *AAS*, *117*, 161.
Hamaker, J.P., 1998, *AAS*, in press.

What's New

Athol Kemball and Tim Cornwell - NRAO, Socorro

- The AIPS++ Web page layout and content has been revised to incorporate user comments and new features. These include changes to list *Glish* information on a separate page, a new page for vacant positions and substantial revisions to the *Getting Started* page, including new recipes and documentation. In addition, the personnel page has been updated to reflect current information.

- The single-dish analysis package *dish* has been significantly revised in this period. Changes include a major revision of the averager to support multiple Stokes, and more general treatment of the x-axis velocity coordinates. Multiple Stokes support is also available for baseline-fitting, smoothing and re-gridding. Selection is now a true operation, and the action button has been renamed as Apply throughout for uniformity. Infrastructure changes include improvements to printing capabilities and the use of Measures in Doppler conversion calculations.

- Work has proceeded on the GBT utilities in support of GBT commissioning. A utility has been provided to read initial data from the GBT spectrometer into AIPS++. Other changes include revisions relevant to RFI issues, and the application of Tcal values properly to spectral processor data. The script *gbt.g* has been added as an interface to the commissioning tools.

- A utility has been added to convert Parkes Measurement Sets (in multibeam format) into AIPS++ images.

- The initial implementation of the algorithm/applicator framework for parallelization has been checked-in, including the transport layer, which handles parallel communications transport mechanisms (e.g., MPI). A implementation of a parallel Clark CLEAN using this methodology has also been added to the system.

- There have been various revisions to the WSRT data display tools, including a new script, *msbrick.g*, allowing tiled access to uv-data.

- The Measures system has been revised to include full support for MBaseline, Muvw, and MEarthMagnetic classes in the Measure holder system, and parameter implementations. Further changes related to the split of Quanta from Measures have also been made, and the Measures test programs have been updated. In addition, code to support foreign parameters has been added.

- system for translating the VLA calibrator list into an AIPS++ table has been added.

- The data display utility, `visplot.g`, has been revised to incorporate new features or bug fixes suggested by users. These include support for `u` vs `¡v` in generic plots, more information in the pop-up windows, and improved help information and display of `z`-axis values.

- The catalog script, `catalog.g`, has been similarly revised to incorporate user feedback. Changes include the ability to view text files, create files, the addition of an edit button and removal of the import function.

- The Sky module has been renamed to Imager, and the cal module to Calibrater, in order to improve understanding of their purpose and to remove a naming conflict with the UNIX calendar utility "cal".

- The utilities *polyfitter.g* and *sinusiodfitter.g* have been split off from *numerics.g*. In addition, *lsfit.g* and *matrix.g* have been added to *mathematics.g*.

- An updated BIMA filler has been checked-in, including a GUI and a script to launch it from *Glish*.

- The **RPFITS** code has been revised to support the new conventions for Y2000 date formats.

- A new *Glish* function `as_evalstr()` has been added to convert values into a string that can be used with `eval()`.

- New terminology has been adopted for object-oriented concepts in the user documentation. This will be used as widely as possible in the future, and includes the use of terms object, function for `method`, and `global function`.

- There is now added support for the Hierarchical Data System (HDS) data format, including **C++** wrappers for the HDS library routines.

- A variety of minor changes have been made to support syntax requirements of the **egcs** (**C++**) compiler.

- New AIPS++ notes include a revised *System plan* (220), and *How to make an* AIPS++ *release* (221)

- Capabilities supporting region selection from images are now available using the region manager (*regionmanager.g*), and associated GUI.

- The synthesis script *simpleimage.g* has been revised to allow selection by field, the display of the underlying *Glish* being used, and to correct a bug in setting the phase center.

- An initial implementation of wide-field imaging has been added to the synthesis module, *imager.g*. This will become available as a system suitable for users in subsequent releases.

- A new AIPS++ logo has been adopted for general use within the project, shown above.