*Astronomical Data Analysis Software and Systems V*
*ASP Conference Series, Vol. 101, 1996*
*G. Jacoby and J. Barnes, eds.*

# Programming in Glish

Darrell R. Schiebel

*National Radio Astronomy Observatory*[1], *520 Edgemont Road, Charlottesville, VA 22903*

**Abstract.** This paper describes the Glish[2] programming language. Glish is a dynamically typed, array oriented language which also supports distributed clients. Glish has integral, floating point, string, complex, and heterogeneous record values. Any of these values may be used as part of the interpreter to client communications. Both synchronous and asynchronous client interaction are supported. The AIPS++ project (Glendenning 1996) at the NRAO[3] is using Glish as the basic command line interface and as the backbone for internal control and communication. Glish may also be used for communication between AIPS++ and telescope control systems; this is the case with the Green Bank Telescope (Garwood 1996).

## 1. Introduction

Glish is an interpreted array oriented language. Whole array arithmetic is an important factor in simplifying users' interactions with data. In this respect, Glish's syntax and semantics are loosely based on "S" (Becker, Chambers, & Wilks 1988). To this data manipulation language, Glish also adds distributed clients. One of the key features of Glish's client interface is its ability to handle the events (messages) from multiple clients in an asynchronous manner. This allows a Glish script to start several clients and specify how events should flow between them irrespective of event ordering.

AIPS++ selected Glish after some explorations into creating an array oriented language and a distributed object system from scratch. It quickly became apparent that creating a distributed object system which is integrated into a language is not a simple task. Happily, however, Glish filled these gaps very nicely. As a result, Glish is a cornerstone of AIPS++.

Glish was primarily developed at the Lawrence Berkeley Laboratory (LBL) (Paxson & Saltmarsh 1993) and much of the initial development was funded by the Superconducting Super Collider project. Glish was designed with control

[1] The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

[2] http://www.cv.nrao.edu/glish/

[3] http://www.nrao.edu/

```
 a := 11:18                # create an 8 element vector
 a[ 3:6 ] := 7:10          # reassign the middle 4 elements
 a[ [1,8] ] *:= 2          # double the extreme elements
 a[ a < 9 ] := 0           # zero elements less than 9
 print a
[22 12 0 0 9 10 17 36]
 b := array(1:16, 4, 4)    # create a 4x4 matrix initialized
                           #     to 1:16
 b[1:2, ] *:= 2            # double the first two rows
 b[ array(1:4,4,2) ] := 1 # set the diagonal elements to 1
 b[ b > 20 ] := 0          # zero elements greater than 20
 print b
[[1:4,]
   1 10 18 0
   4 1  20 0
   3 7  1  15
   4 8  12 1]
```

Figure 1.    Array and Vector Manipulations

systems in mind, and as a result, the event handling in Glish works well. Most of the AIPS++ extensions have been in the area of the data manipulation language.

## 2. Array Orientation

With few exceptions, values in Glish are vectors. All numeric operations are performed on vectors. In fact, multidimensional arrays are implemented as vectors with extra information, i.e. the shape of the array. Functions which operate on vectors element by element, thus, work for both arrays and vectors.

Array oriented languages allow the user to operate on sub-arrays of a larger array, and for this, Glish has a flexible format for taking *slices* (contiguous sub-arrays) and *picks* (non-contiguous elements) of arrays and vectors. Figure 1 shows how array access within Glish works.

In this figure, vectors are created with the sequence operator, ":". This operator creates a vector initialized to the specified sequence. For example, `3:5` would create a vector containing `[3 4 5]` , and `1:-1` would yield `[1 0 -1]` . Arrays are created with the `array` function. The first parameter to this function is the initialization values for the array. This initialization vector is repeated if its length is less than the length of the array. The final parameters to `array` are the lengths of each of the array's dimensions. Figure 1 also has examples of how boolean vectors can be used to mask portions of the array, e.g. `a[a < 9] := 0` .

```
# set up the URLs to display
pages := [ 'http://www.cv.nrao.edu/', 'http://www.gb.nrao.edu/',
           'http://fits.nrao.edu/' ]
page := len(pages)                   # set current page
mosaic := client("glish mosaic.g")   # start mosaic script client
timer := client("timer")             # start timer client
whenever timer->ready do {           # with each timer event
    page := page % len(pages) + 1    # update the URL number
    send mosaic->move( pages[page] ) # send event to move mosaic
}
send timer->interval(40)             # set the timer interval
```

Figure 2. Slideshow with Glish and Mosaic

## 3. Distributed Clients

Distributed clients are useful to users and programmers because they permit distribution and localization. The usefulness of distribution is obvious; work can be selectively distributed across machines on a network. Localization, however, is perhaps the biggest win. By having clients which are loosely coupled to the interpreter, pieces of the system are localized and isolated. The only interaction among the pieces of the system is a defined protocol of events. The interaction between the interpreter and the clients must, however, be sufficiently rich. In the case of Glish, nearly all Glish values can be passed between the interpreter and its clients. All of this means that for programmers portions of the system can be developed with little regard for the rest of the system, and for users it means that portions of their environment can be updated without disturbing them and without recompiling the whole environment.

In Glish, the interpreter is the hub through which events flow, and a Glish script directs how the events flow from one client to others. In many ways, Glish clients and events correspond to objects and messages in some object oriented systems. Event flow in Glish is directed primarily with the `whenever` statement. Figure 2 shows how clients are created and how `whenever` statements are used. In this example, two clients are used — `timer`, which sends an event periodically with a specified interval, and `mosaic`, which is a Glish script client (see Figure 3). The code in Figure 2 initializes a string array, `pages`, with the URLs which we want to loop through. It then creates the two clients with the `client` function. The `whenever` statement points mosaic at a new URL each time `timer` generates a `ready` event. The final line initializes `timer` to generate events 40 seconds apart. This simple example cycles mosaic through a series of URLs.

Figure 3 shows the Glish script client `mosaic.g`. A "script client" is a Glish script which behaves like a client for another Glish interpreter. This client in particular handles moving Mosaic from one URL to another. In script clients, the `script` value is the source of events to the client and the sink for events generated by the client.

```
## start mosaic
mosaic := shell("Mosaic>/dev/null 2>&1",async=T)
## get mosaic's PID from "ps"
p := shell('(ps -xww|egrep -v "sh|grep"|grep Mosaic)||echo')
pid := split(p)[1]                        # get PID from the line
file := paste("/tmp/Mosaic.",pid,sep='')# build mosaic file name
whenever script->move do {               # with each move event
    shell(paste("(echo goto;echo '",$value,"')>",file))
    shell(paste('kill -USR1',pid))       # move mosaic to new URL
}
```

Figure 3. Steering Mosaic (mosaic.g)

## 4. Conclusion

This paper has only addressed the most interesting aspects of Glish. The aspects of Glish which one now expects from modern programming languages, e.g. functions, structures, control loops, conditionals, etc., have not been addressed. The implementation details were also not addressed. For this and other information, I refer you to the Glish home page found at the URL given at the beginning of this paper.

The topics which this paper does address, however, demonstrate some of the most interesting aspects of Glish. Indeed, it is primarily for these reasons that AIPS++ has adopted Glish. While we have continued to improve Glish, much gratitude is owed those who laid Glish's foundation at LBL.

The Glish additions which are currently being explored at the NRAO include — a Tk binding for Glish which allows individual Tk widgets to behave like Glish clients, improved security for Glish clients, a module system for Glish, clients which look more like distributed objects, and the ability for clients to extend the Glish language itself.

## References

Becker, R. A., Chambers, J. M., Wilks, A. R. 1988, The New S Language (Pacific Grove, Wadsworth & Brooks)

Paxson, V., Saltmarsh, C. 1993, Proceedings of the Winter Technical Conference (Berkley, USENIX Assoc), p.141

Glendenning, B. E. 1996, in ASP Conf. Ser., Vol. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes (San Francisco: ASP)

Garwood, R. W. 1996, in ASP Conf. Ser., Vol. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes (San Francisco: ASP)