NOTE 257 –

Creating a Useful Glish Client (C), Glen Langston, NRAO-GB

2003 April 22

May 29, 2015

1 Summary

C programs are easily added to the glish command prompt in AIPS++, by creating glish clients. We present an example client which calculates radio source flux densities based on the Ott *et al.* 1994 and Baars *et al.* 1987 models for a set of reference sources. This example is intended for C programmers who wish to add new capabilities to glish and AIPS++. This example can be modified for "easy" incorporation of stand alone functions.

This document describes the components needed to create a glish client. Two types of interfaces between C and glish are shown, *1)* passing strings and *2)* arbitrary sized double precision arrays.

The steps for construction, testing and execution are given. In appendices, the glish and C++ code is listed. To view the entire C code functions, down-load the tar file from the web. This document and the code are on the web at: http://www.gb.nrao.edu/∼glangsto/aips++/glishClient

This document is an update of the examples created by Rick Fisher in 1997. Rick's document is useful as an example of how any type of Glish variable can be passed between glish, C, Fortran and C++. See: http://www.gb.nrao.edu/∼rfisher

2 Components

These Glish clients are run as a part of an AIPS++, DISH or vanilla glish session. The clients are produced from a number of components listed below:

ottClient.g Glish scripts interfacing to the C++ wrapper around the observers C program. This script loads the glish procedure. The scripts handle data typing and runs a test. A Unix shell interface to the program `ott` is presented. session to confirm the configuration.

`ottClient.cc` C++ wrapper that identifies Glish events and calls the C functions.

`ottFluxes.c` C function to calculate the Ott *et al.*1994 source model flux densities.

`ott.c` Stand alone C function to calculate the Ott *et al.*1994 model. This function also used ottFluxes.c

`MakeOtt` Make file that is configured for the local AIPS++ installation. This file will probably require modification for successful installation of the ottClient

3 Test Execution of `ott`

After creating the client and stand-alone program, test `ott` with different inputs. With only the frequency (MHz) argument, `ott` program prints the flux densities for all sources for a input frequency (MHz). Output flux densities are in Janskies.

```
% ott 1408
 Calculating flux densities for frequency 1408.000000 MHz
    CASA   2092.0688+/-  160.548
   3C286     14.6208+/-    0.940
   3C48      16.3005+/-    1.192
   3C147     22.0340+/-    1.384
   3C138      8.4819+/-    0.848
1934-638     16.2053+/-    1.621
   3C405   2706.6796+/-  429.630
    DR21     26.6606+/-    3.627
 NGC7027      7.9449+/-    3.972
   3C295     22.2488+/-    1.676
   3C123     47.3906+/-    4.739
   3C161     18.6043+/-    1.860
   3C218     42.7130+/-    4.271
   3C227      7.6057+/-    0.380
 3C249.1      2.2779+/-    0.228
```

```
    VIRA    203.1464+/-   10.157
 3C309.1      7.4350+/-    0.520
   3C348     46.9291+/-    2.346
   3C353     56.3030+/-    2.815
              1.0000+/-    1.000
Epoch 2002DEC02 Cas-A Flux density based on   22.92 year decrease
Cas-A decrease factor:     0.9907/yr =>     0.8081 total factor
    CASA   1690.5828+/-  129.737
```

The ott program will also calculate the flux density for a single source at a single frequency. This capability is used in ottClient.g, function ottShell() to call a C program from glish.

```
% ott -s 3C286 1408
 14.6208
```

4 Construction

To create ott, the stand alone program, and ottClient, the glish client, down-load the TAR file ott.tar. Extract the components, edit the MakeOtt makefile and create the program in the following steps.

```
% mkdir ottClient                    # create a new directory
% cd ottClient                       # go to the directory
% wget http://www.gb.nrao.edu/~glangsto/aips++/ott.tar  #use wget or ftp
% tar -xvf ott.tar                   # separate the components
% emacs MakeOtt                      # edit the makefile for local paths
% source /home/aips++/stable/aipsinit.sh   # setup the aips environment
% make                               # create the programs
```

The tar-ball ott.tar also contains the executables for RedHat Linux. The make step can be skipped if ott works properly at the command line prompt. The individual components of ott.tar are at: http://www.gb.nrao.edu/*sim*glangsto/aips

5 Execution of ottClient.g

The first step in using the C clients in glish is loading the programs and glish scripts into the local aips++ directory.

Next, the shell environment variables must be configured for the local aips++ installation. This is done with a Unix script. Consult with your aips++ expert to find the script.

The client is tested during loading in the glish session. An example session is:

```
% bash                              # start the bash shell
% source /home/aips++/stable/aipsinit.sh
AIPS++ Version 1.8 Build 321
% glish -l ottClient.g
reading .glishrc_local from /home/aips++/stable2
```

```
Test Execution of Ott functions:
ottSourceName ( ’3C295’)          := T
freqs                             := [2e+08 3e+09 4e+10 5e+11]  (Hz)
ottFluxes( freqs)                 := [76.2198 11.0079 0.379414 0.00340961]  (Jy)
ottShell( ’3C295’, freqs)         := [76.2198 11.0079 0.379414 0.00341]  (Jy)
Glish version 2.7.
- exit
```

The function ottShell() works fine for a few flux density values. However its performance is too slow for an array larger than a thousand or so frequencies. In this case the function ottFluxes() is far superior.

6 Documentation

The program `ott` prints help if no input arguments are provided, as show below:

```
% ott

ott: prints Ott et al 1994 and Baars et al 1977 flux densities.
usage: ott [-s <sourceName>] [-d <epoch>] frequencyMHz
where [-s <sourceName] is the optional source name.
where <frequencyMHz>   frequency for model flux densities
where [<epoch>]  optional date string for calculating Cas-A decay
                 string is in YYmmmDD format (ie 99feb04 or 05apr01)

The models of Ott, M. Witzel, A., Quirrenbach, A., Kirchbaum, T.P.
Standke, K., J., Schalinski, C. J., and Hummel, C.A.,
1994 Astronomy and Astrophysics, Vol 284 pg 331 and
Baars, Genzel, Pauliny-Toth and Witzel 1977, Astronomy and
Astrophysics, Vol 61, page 99 are used.

The Ott et al models are good for 1408 to 23000 MHz for most sources
For DR21, the Baars et al 1977 model is used.
For Cas-A, the current date is used to calculate flux density
Based on the Baars epoch 1980 model for decline in flux densities
Cas-A Model: 0.97+/.04 - 0.30+/-.04 log(freq./GHz) percent decrease/year

The Error estimates are approximate, using values in the text and
```

```
interpolating.  The default is 5 percent.
Sources Modeled:
      CASA      111-2   2323+588
     3C286   1328+307   1331+305
     3C48    0134+329   0137+331
     3C147   0538+498   0542+498
     3C138   0518+165   0521+166
  1934-638
     3C405       CYGA   1957+406
      DR21   2037+421   2039+423
   NGC7027   2105+420   2107+422
     3C295   1409+524   1411+522
     3C123   0433+295   0437+296
     3C161   0624-058   0627-058
     3C218   0915-119   0918-120
     3C227   0945+077   0947+074
   3C249.1   1100+772   1104+769
      VIRA   1228+127   1230+123
   3C309.1   1458+718   1459+716
     3C348   1648+051   1651+049
     3C353   1717-009   1720-009
```

7 References

1 Baars, Genzel, Pauliny-Toth and Witzel (1977), Astronomy and Astrophysics, Vol. 61, pg. 99.

2 Ott, M. Witzel, A., Quirrenbach, A., Kirchbaum, T.P. Standke, K., J., Schalinski, C. J., and Hummel, C.A., (1994) Astronomy and Astrophysics, Vol. 284, pg 331.

8 Appendix: `ottClient.g`

Below is a listing of the ottClient glish script used to interface glish to the C functions.

```
#File ottClient.g, version 1.3, released  02/12/02 at 11:26:08
#   retrieved by SCCS 02/12/02 at 11:26:18
#Glish event wrapper functions for calling C functions from Glish
#HISTORY
```

```
# 021202 GIL change ottFlux to ottShell
# 021122 GIL minor initial version based on ex_client.g
# 021115 GIL initial version based on ex_client.g
# 021114 GIL update for minor changes to glish
# 970803 JRF Initial version very well documented at
#            http://www.gb.nrao.edu/~rfisher/Glish/ex_client.html

global ottSource := '3C286'

ottFluxes := function ( valu )
# ottFluxes takes an array of frequencies (Hz) and returns an array of
# flux densities (Jy).  Must first set the source name with ottSourceName()
{
    freqFluxes := as_double( valu)              #/* transfer frequencies */
    oFluxes := sf->ottSetFlux( freqFluxes);

    return (oFluxes)
} #/* end of ottFluxes() */

ottSourceName := function ( sourceName )
{ #store source name as global with error checking
  # INPUTS:  sourceName      string name of source "CASA" or "3C286" etc

  global ottSource;

  ottSource := as_string( sourceName);

  global sf := client('ottClient');
  dummyValue := sf->ottSetSource( ottSource);  #/* now set the source name */

  return(T)
} #/* end of ottSourceName() */

ottShell := function ( sourceName, frequencyVector)
{ # ottFlux gets a single ott et al flux via a commandline interface to ott.
  # INPUTS:  sourceName      string name of source "CASA" or "3C286" etc
  #          frequencyVector Array of frequencies (Hz)
  # OUTPUT:  fluxVector      Array of flux densities (Jy)
  # This function is 100 times slower than ottFluxes()
  frequencyShape  := shape( frequencyVector);

  if ( length( frequencyShape) > 1) {
    print "Frequency vector must be one dimensional!"
    return 0;
  }
  n := frequencyShape[1];                  # get number of frequencies
  fluxVector := array( 0, n);              # create output array

  frequencyMHz := as_double( frequencyVector) * 0.000001; #from Hz to MHz */
  for (i in (1:n)) {               # for all frequencies
    shellString := sprintf( "ott -s %s %f", sourceName, frequencyMHz[i]);
    # execute the shell string and convert to double
    fluxVector[i] := as_double( shell( shellString));
  } # end for all frequencies
  return fluxVector;
} # end of ottShell

freqs := [ 2e8, 3e9, 4e10, 5e11];          # set the test frequencies

print ''
print 'Test Execution of Ott functions:'
print 'ottSourceName ( \'3C295\')          :=', ottSourceName( '3C295');
print 'Freqs                             :=', freqs, "(Hz)";
print 'ottFluxes( freqs)                 :=', ottFluxes( freqs), "(Jy)";
print 'ottShell( \'3C295\', freqs)         :=', ottShell( '3C295', freqs),"(Jy)";
```

9 Appendix: ottClient.cc

The C++ wrapper main program to the C functions is listed below. The functions performing the calculations are in ottFluxes.c.

```
/* File ottClient.cc, version 1.2, released  02/12/02 at 11:42:52
   retrieved by SCCS 02/12/02 at 11:43:21

C++ client code wrapper for C functions to calculate the Ott et al fluxes.

HISTORY
021202 GIL remove un-used segments of the code.
```

```
021118 GIL add some comments
021115 GIL Initial version based on ex_client.cc
021114 GIL Removed support for complex functions
970803 JRF Initial version, well documented at
             http://www.gb.nrao.edu/˜rfisher/Glish/ex_client.html
DESCRIPTION
Rick Fisher created a very nice set of example code that allows an
observer to quickly create a interface from C to glish.
This function impliments the C++ wrapper to a simple C program to calculate
the Ott et al 1994 flux densities for a set of reference sources.

There are two steps in the process:  First is setting the source name
for which the values are calculated.  The second step is providing
an array of frequencies (Hz), for which the flux densities are required.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "Glish/Client.h"

// Declare all of the C/C++ functions that you are going to use or put
// the main() function at the end.
void setFluxes(Client &c, GlishEvent *e);
void setSource(Client &c, GlishEvent *e);

extern "C" {
  char * setOttSourceName( char * source);
  char * setOttSourceFluxes( long n, double values[]);
} /* end of C declares */

int main (int argc, char **argv) {
    // This creates a required client object.
    Client c(argc, argv);
    // The client can be invoked with arguments, but we‘ll bypass that
    // complication.
    if (argc > 1) {                     /* if any argument, explain usage */
        printf ("Usage: cl := client(’ex_client’)\n");
        return 1;
    } /* end if an argument */
    // Create a pointer to be assigned to a received glish event.
    GlishEvent *e;

    // Stay in this loop until the client is terminated.  The c.NextEvent()
    // function blocks until it receives an event from glish.  It then
    // returns an event pointer that is used to access the values passed
    // from glish.
    while ((e = c.NextEvent())) {
        // Search for an expected event name and execute the appropriate
        // function when found.
// vvvvvvvvvvvv Your code substituted below here. vvvvvvvvvvvv
        if (!strcmp(e->Name(), "ottSetSource")) {
  setSource( c, e);
        } else if (!strcmp(e->Name(), "ottSetFlux")) {
          setFluxes( c, e);
// ^^^^^^^^^^^^ Your code substituted above here. ^^^^^^^^^^^^
        } else {
            // Report an error if an event name is not recognized.
            c.Unrecognized();
        }
    }
    return 0;
} /* end of client main() */

void setFluxes(Client &c, GlishEvent *e)
{
    // See do_int_squared() function for comments on statements which
    // are common to all functions.
    Value *val = e->Val();
    long array_length = val->Length(), i = 0;
    double *return_value = new double[array_length];

    for (i = 0; i < array_length; i++)     /* default value is 1. */
        return_value[i] = 1.0;

    if (val->Type() != TYPE_DOUBLE || array_length <= 1) {
        printf("Double type array expected from ‘setFluxes’\n");
    } else {
        double * dataValues = val->DoublePtr();
        double * freqFluxes = new double[array_length];
// vvvvvvvvvvvv Your code substituted below here. vvvvvvvvvvvv
        for (i = 0; i < array_length; i++)
          freqFluxes[i] = dataValues[i];

        setOttSourceFluxes( array_length, freqFluxes);
        for (i = 0; i < array_length; i++)  /* transfer out */
          return_value[i] = freqFluxes[i];
// ^^^^^^^^^^^^ Your code substituted above here. ^^^^^^^^^^^^
        delete freqFluxes;
    } /* end else if data of expected type */
    Value *rep = new Value(return_value, array_length, COPY_ARRAY);
    c.Reply(rep);
    delete return_value;
    Unref(rep);
} /* end of setFluxes() */

#define MAXSIZE 100                   /* set string max size */
void setSource(Client &c, GlishEvent *e) {
    // setSource() takes the source name event and transfers the name to the C
    // functions.
    Value *val = e->Val();
    char return_value[MAXSIZE] = "", sourceName[MAXSIZE] = "", * errMsg = ’\0’;

    if (val->Type() != TYPE_STRING) {
        printf("String type value expected from ‘reverse_string’\n");
    } else {
        char *received_value = val->StringVal();
        strncpy( sourceName, received_value, MAXSIZE);
        sourceName[MAXSIZE-1] = ’\0’;
        // The call of val->StringVal() implicitly allocates memory
        // for the string which must be deleted to avoid a memory leak.
        delete received_value;
// vvvvvvvvvvvv Your code substituted below here. vvvvvvvvvvvv
        errMsg = setOttSourceName( sourceName);
        if (errMsg != ’\0’)                /* check for error strings */
  strcpy( return_value, "!!!! ");
strcat( return_value, sourceName);
// ^^^^^^^^^^^^ Your code substituted above here. ^^^^^^^^^^^^
    }
    Value *rep = new Value(return_value);
    c.Reply(rep);
    Unref(rep);
} /* end of setSource() */
```

10 Appendix: MakeOtt

The makefile, `MakeOtt` used to create the clients is listed below. The tar-ball `ott.tar` contains a symbolic link from `MakeOtt` to `makefile`.

```
#Make file for C++ wrapper to C code for Ott et al 1994 flux densities.
#HISTORY
# 021122 GIL Add make file for stand alone program ott
# 021118 GIL Update for configuration changes in GB.
# 970101 JRF Initial version
#DESCRIPTION
#This make file creates two programs
#ottClient: glish client allowing a glish session to call C functions
#ott:       stand alone program (linux or solaris) calc Ott etal

# at the ATNF
#BASE_DIR = /aips++
# at Green Bank
#BASE_DIR = /aips++/test
BASE_DIR = /home/aips++/stable

#Now use Linux  not solaris
#LIBRARIES = -L$(BASE_DIR)/sun4sol_gnu/lib
LIBRARIES = -L$(BASE_DIR)/linux/lib

INCLUDES = -I$(BASE_DIR)/code/aips/glish/include -I./xlib

#this make file makes two programs
EXECUTABLES = ottClient ott

all: $(EXECUTABLES)

#selected code taken un-modified from the VLBA antenna control code and
#OVLBI tracking station.
VLBASTRINGS = str2mjd.o str2rad.o  stripWhite.o time2str.o mjd2str.o \
srclist.o cvrtuc.o dateObs2DMjd.o today2mjd.o ottFluxes.o
```

```
#the VLBA code environment includes definitions of C structures not
#used here.  This stub replaces the include with the standard definitions
vlb.h :
ln -s STDDEFS.H vlb.h

#The VLBA code contains two great general purpose includes of constant values
#Math constants and the definitions of TRUE, FALSE etc.
$(VLBASTRINGS) : MATHCNST.H STDDEFS.H vlb.h $(@:.o=.c)

#now describe how to compile
CC=gcc
CFLAGS= -O -Wall -g $(INCLUDES) $(LIBRARIES)

COMPILE.c= $(CC) $(CFLAGS) $(CPPFLAGS) -c

#rule for converting .c to a .o file
.c.o :
@ $(RM) $@
$(COMPILE.c) $<

ottClient.o : ottClient.cc
g++ $(INCLUDES) -c ottClient.cc

ottClient : ottClient.o $(VLBASTRINGS) $(OTTCLIENTC)
g++ -o ottClient $(OTTCLIENTC) $(VLBASTRINGS) ottClient.o \
$(LIBRARIES) -lglish -lsos -lnpd -lm

ott : ott.o $(VLBASTRINGS) $(OTTCLIENTC)
$(CC) $(CFLAGS) -o ott $(OTTCLIENTC) $(VLBASTRINGS) ott.o -lm
```