Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

The SLANG/CIAO Synergy: Using S-Lang within CIAO

G. Germain, R. Milaszewski, W. McLaughlin, J. Miller

Harvard Smithsonian Center for Astrophysics
60 Garden Street, Cambridge, MA. 02138

Abstract. The integration of S-Lang into the Chandra Interactive Analysis of Observations (CIAO) infrastructure has transformed the capabilities of CIAO. There are several ways to use S-Lang. One is to write a S-Lang function which can be called from C/C++, CIAO applications, or any S-Lang prompt. Another is to write a C/C++ program which is made into a S-Lang intrinsic, allowing it to be called from any S-Lang script. The key element is that a C/C++ or CIAO application can call a S-Lang function/intrinsic, and that a S-Lang function/intrinsic can call a C/C++ or CIAO application. To use this capability, data must be exchanged between the C/C++ space and S-Lang space. This paper describes some of the mechanisms available for that data exchange. These mechanisms are illustrated through simple C/C++ and S-Lang program pairs, the S-Lang intrinsic methodology used by the CIAO S-Lang function "univar", and the use of the VARMM library by CIAO functions: Chandra Imaging and Plotting Software (ChIPS) UNIVAR, and the Graphical File Browser (PRISM) "histogram".

1. Introduction

S-Lang is an interpreted language created by John Davis of the Center for Space Research at MIT[1]. The S-Lang scripting language is embedded into CIAO, and is used in several CIAO applications such as ChIPS and PRISM. It is available to CIAO users and developers for execution of their own S-Lang scripts, or scripts contributed from other organizations. In addition, there is a large body of analysis functions written in C or C++ which can be used for data analysis. For example, CIAO provides various functions which are used in CIAO, and are available to developers and users. Two examples of these are the "univar" and "histogram" functions. Once a user or developer acquires a facility with scripting, and utilizing existing functions in those scripts, they acquire great analytical power. With these techniques in hand, each language can be used when and where they would be most beneficial. But to acquire this capability, users and developers must know how to exchange information between C/C++ and S-Lang, and call functions in one language from another. This paper describes several methods available for that data exchange.

[1]`http://www.s-lang.org`

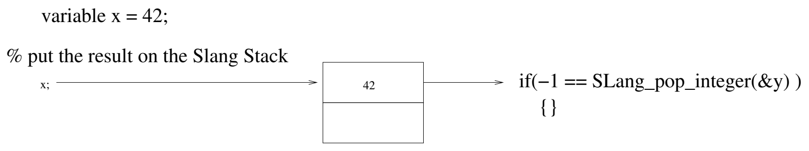

variable x = 42;
% put the result on the Slang Stack
x;
42
if(−1 == SLang_pop_integer(&y))
{}

Figure 1. S-Lang to C value exchange

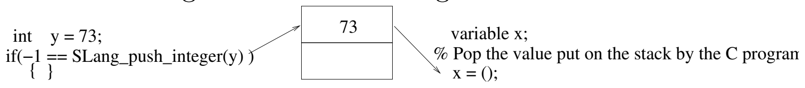

int y = 73;
if(−1 == SLang_push_integer(y))
{ }
73
variable x;
% Pop the value put on the stack by the C progran
x = ();

Figure 2. C to S-Lang value exchange

2. Simple Exchanges

S-Lang is a stack based language. One must learn how to make use of the stack, to a much greater degree than in C or C++, in order to do serious S-Lang programming. For example, exchanging values between a C program and a S-Lang script is done using the stack. To push an integer variable (e.g. "x") onto the stack, in S-Lang, the user stipulates the variable followed by a semi-colon: "x;" (Figure 1). To capture that value in a C program, the user pops the value off the stack with the "SLang_pop_integer" function. This works regardless of data structure type. If "x" is an array of 10 floats, then "x"; pushes the 10 values on the stack.

In Figure 2, the C program pushes an integer value onto the S-Lang stack, using the "SLang_push_integer" function. The S-Lang program on the right, pops it off with the S-Lang "()" syntax. In S-Lang, "()=" pops a data structure off the stack and discards it. "x = ();" pops a data structure off the stack and assigns it to the S-Lang space variable, "x".

In addition to exchanging values via the stack, C programmers can create and manipulate S-Lang variables directly:

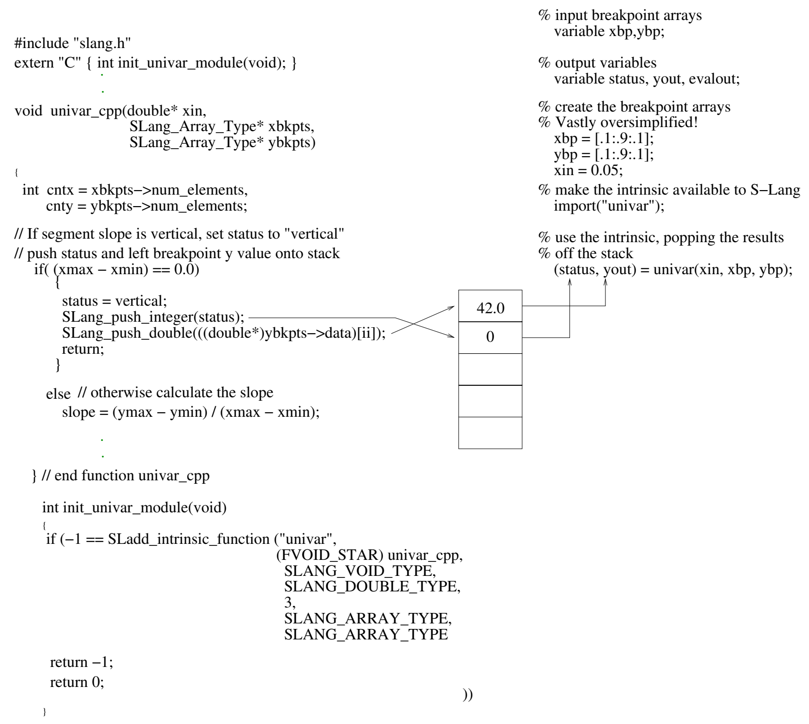
```
// Tell S-Lang to create the variable, x, and initialize  it to 42
   if(-1 == SLang_load_string(‘‘variable x = 42;’’))  {......}
```

There are two ways a user can run the programs and scripts which exchange data. One method requires two windows: one in which the C program is run, and another with a S-Lang prompt. Alternatively, a C program can execute a S-Lang script using the "SLang_load_file" function.

These are the simplest methods of exchanging data; all subsequent methodologies are based upon these basic operations. The necessary C/C++ functions are found in "slang.h". These are the methods to be used when minimal coupling between scripts and functions is required.

3. C/C++ Intrinsics

S-Lang has many strengths, such as array manipulations. But for heavy mathematical processing C or C++ is a better choice than an interpreted scripting language. S-Lang provides a means to incorporate functionality best written in



```
#include "slang.h"
extern "C" { int init_univar_module(void); }
                  .
                  .

void  univar_cpp(double* xin,
                        SLang_Array_Type* xbkpts,
                        SLang_Array_Type* ybkpts)

{
  int  cntx = xbkpts−>num_elements,
       cnty = ybkpts−>num_elements;

// If segment slope is vertical, set status to "vertical"
// push status and left breakpoint y value onto stack
    if( (xmax − xmin) == 0.0)
      {
        status = vertical;
        SLang_push_integer(status);
        SLang_push_double(((double*)ybkpts−>data)[ii]);
        return;
      }

    else  // otherwise calculate the slope
        slope = (ymax − ymin) / (xmax − xmin);

                  .
                  .

   } // end function univar_cpp

    int init_univar_module(void)
    {
     if (−1 == SLadd_intrinsic_function ("univar",
                                       (FVOID_STAR) univar_cpp,
                                         SLANG_VOID_TYPE,
                                         SLANG_DOUBLE_TYPE,
                                         3,
                                         SLANG_ARRAY_TYPE,
                                         SLANG_ARRAY_TYPE
                                                                    ))
      return −1;
      return 0;
    }
```

```
% input breakpoint arrays
   variable xbp,ybp;

% output variables
   variable status, yout, evalout;

% create the breakpoint arrays
% Vastly oversimplified!
   xbp = [.1:.9:.1];
   ybp = [.1:.9:.1];
   xin = 0.05;
% make the intrinsic available to S−Lang
   import("univar");

% use the intrinsic, popping the results
% off the stack
   (status, yout) = univar(xin, xbp, ybp);
```

Figure 3. The C Intrinsic Method

C, into a S-Lang script. To do this, the user casts the C function into a S-Lang intrinsic, making it just as available to the user as "sin(x)".

Figure 3 illustrates the method used to make and use a C++-based S-Lang intrinsic. On the left is the CIAO function "univar"; written in C++, built as a module, and made into a S-Lang intrinsic. It is available for direct use in any S-Lang script the CIAO user wishes to write.

To make "univar" known to S-Lang, the user must supply the function "init_univar_module". S-Lang calls the function automatically. "init_univar_module" contains the key function call which adds the intrinsic: "SLadd_intrinsic_function".

The S-Lang script on the right, in Figure 3, shows how a S-Lang intrinsic is used. The S-Lang programmer must first import the module, as shown. When called, the user passes in an x input (xin), an array of x breakpoints, and an array of corresponding y breakpoints (xbp, ybp). "univar" returns the result by pushing a status value indicating the mathematical validity of the result, and the interpolated value, onto the S-Lang stack. The S-Lang script captures the resultant values off the stack using the "(status, yout)" stack popping syntax, as shown in the script.

4. Application Use of Embedded Libraries.

Writers of S-Lang and S-Lang/C++ scripts often need common functionality such as FITS or ASCII file operations and data structure representation. In the previous examples, S-Lang variables that are created and are to map to corresponding C++ variables, must be kept synchronized by the programmer. To simplify S-Lang/C++ integration, CIAO provides the programmer with VARMM: a library of classes and methods which is a data interface between C++ and S-Lang (S. Doe, et. al. 2000). VARMM provides classes for data representation, methods to manipulate those data structures, FITS and ASCII file operations, and a simple API. VARMM data elements can be linked to their S-Lang variable counterparts so that a change in the value of the S-Lang variable changes the value in the VARMM object. VARMM ASCII or FITS file read operations can create VARMM and S-Lang data structures containing the file input values.

For a brief illustration of how VARMM can be used, we choose a CIAO C++ application, PRISM, which is commanded to plot a histogram of a selected column of a FITS file. PRISM invokes ChIPS and commands ChIPS to forward the following statement to the S-Lang parser:

prism_data = readbintab("pe.fits[2][cols pha]");

"readbintab" is a VARMM library method which opens the FITS file, pe.fits, creates the S-Lang data structure, "prism_data", extracts the "pha" column of data, and stores the data in "prism_data". PRISM will send a subsequent command to ChIPS to execute the S-Lang script, "histogram", which results in the histogram plot. "prism_data", a variable in S-Lang scope, is available to the "histogram" script as well as any other script the programmer wishes to write and execute.

5. Conclusion

S-Lang is integrated with CIAO and can be used in conjunction with C or C++ programs to gain enhanced analysis capabilities. Users can select either S-Lang or C/C++; whichever is most appropriate for the application. Using S-Lang and C together requires methods to exchange data between the two scopes, using the S-Lang stack. Simple exchanges can be done using push and pop commands in S-Lang, and, for C, push/pop functions found in "slang.h". C programs can issue S-Lang commands directly or invoke S-Lang scripts. This provides the simplest interface and the least coupling between the script and the C program. C/C++ functions can be integrated into S-Lang scope by making them into a S-Lang intrinsic, allowing the functionality to be distributed. Common functions needed by most programmers, such as file I/O, data structure creation and synchronization, are supplied by VARMM. VARMM also supplies an API which makes integration of S-Lang and C++ much simpler.

Acknowledgments. This work is supported by the Chandra X-Ray Center under NASA contract (NAS8-39073).

References

S. Doe, M. Noble, and R. Smith 2000, Interactive Analysis and Scripting in CIAO 2.0, ADASS X, P2-41

J. E. Davis 2003, S-Lang Library C Programmer's Guide, V1.4.2