IRAF SPP Programming

by the NOAO IRAF Team:

Mike Fitzpatrick

Rob Seaman

Frank Valdes

Nelson Zárate

R. Seaman – 22 July 2003

References

- *"An Introductory User's Guide to IRAF SPP Programming" by Rob Seaman*
- *IRAF package of examples from the text*
- *Quick Reference Card*
- *Document refers to v2.10, but still valid*
- *http://iraf.noao.edu/docs/prog.html*
- *Further references within User's Guide*

IRAF Design Philosophy

- *Extreme portability*
- *Language interface (SPP)*
- *Powerful procedural interfaces (VOS)*
- *Host dependent kernel (OS interface)*
- *Bootstrap utilities (***xc** and **mkpkg***)*
- *CL context and resources*
- *Standards*

IRAF Tasks

- *Tasks live in packages (see p. 55)*
- *Compiled programs (SPP)*
- *Interpreted scripts (CL)*
- *Foreign tasks (Unix and IMFORT)*
- *I/O redirection*
- *Background execution*
- *Host execution*
- *Parameters*

IRAF Tasks (*continued*)

- *Name abbreviation (dictionary, not path)*
- *Graphics*
- *Image display*
- *Cursors*
- *IRAF networking*
- *IRAF environment*
- *Virtual pathnames*
- *External packages*

Hello, world!

```
# HELLO -- Sample program introducing SPP.

task hello = t_hello_world

procedure t_hello_world ()

begin
        call printf ("Hello, world!\n")
end
```

Compiling an IRAF Task

```
cl> xc hello.x
hello.x:
   sys_runtask:
   t_hello_world:
hello.f:
   sysruk:
   thelld:
link:
```

Declaring an IRAF Task

```
cl> task $hello = hello.e
```

or

```
cl> task $hello = home$hello.e
```

or

```
cl> task fibonnaci = home$fibonnaci.e
```

Running an IRAF Task

```
cl> hello
Hello, world!

cl> hello > foo
cl> type foo
Hello, world!

cl> hell &
[1]
cl> Hello, world!
[1] done  0.0 0:00 0%

cl> $hel
Hello, world!
Time (hello) 0.00 0:00 99%
```

SPP Basics

- *IRAF file names* (`.x, .h, .e, .o, .a, ...`)
- ***No semicolons*** *(except for null statements)*
- *Continue with comma, operator or backslash*
- *Free form indentation and blank lines*
- *Comment lines begin with* `#`
- `define` *constants with macros*
- *Declare all variables and external functions*
- *Don't declare intrinsic functions (overloading)*

SPP Basics (*continued*)

- *Start all subroutines and typed functions with* `procedure` *statement*
- *Use* `begin` *and* `end` *within procedures*
- *Reference untyped procedures with* `call`
- *Braces (*`{}`*) surround execution blocks*
- *Arrays are specified with brackets (*`[]`*)*

SPP Conditional Statements

```
if (expression) {
    statements
} else if (another expression) {
    other statements
} else {
    more statements
}

switch (integer expression) {
case integer :
    statements (does not fall through)
case another integer :
    other statements
    break
default:
    yet more statements
}
```

SPP Looping and Iteration

```
do i = 0, 10, 2 {
    statements
}

for (i=1; i <= 10; i=i+1) {  # no ++ or += constructs
    statements
}

while (boolean expression) {
    statements
    next   # not “continue”
}

repeat {
    statements
} until (boolean expression)
```

SPP Branching

```
break                      # terminates conditional or loop
next                       # skips to top of loop

return                     # exits procedure
return (typed value)       # exits function, returns value

define done_ 99            # maximum label for goto is 99
goto done_
        statements
done_
        more statements
```

SPP Declarations

- *Arrays are 1 indexed*
- *Scalar types similar to C (*`real`*, not* `float`*)*
- `pointer` *is an explicit type*
- *Fortran style* `common`
- *Fortran style* `data` *statements*

Include Files

- *Various interfaces require an* `include` *file*
- *System* `include` *files are kept in* `iraf$lib`
- *Key constants preloaded from* `hlib$iraf.h`
- *Machine constants are in* `hlib$mach.h`

CL Parameters

- *Query (prompt the user)*
- *Hidden (provide a default)*
- *Menu mode*
- *Attributes (type, range/enum, prompt, ...)*
- *Parameter editor (***eparam***)*
- *Private* **uparm** *directory (learn/unlearn)*
- *Parameter sets (and package parameters)*
- *Parameter caching*

Tasks with Parameters

fibonnaci.x (see ***page 7*** *of User’s Guide):*

```
nterms = min (clgeti (”nterms”), MAX_TERMS)
```

examples$src/fibonnaci.par:

```
nterms,i,a,,1,50,Number of terms in the...
```

usage:

```
cl> fibonnaci.nterms = 7
cl> lpar fibonnaci
       nterms = 7                Number of terms...

cl> = fib.nterms.p_max
50
```

Tasks with Parameters (*cont.*)

```
cl> fib
Number of terms in the Fibonnaci sequence (1:50) (7): <cr>
 N       Algebraic      Sequence
 1               1      1
 2               1      1
 3               2      2
 4               3      3
 5               5      5
 6               8      8
 7              13      13

cl> fib 3
 N       Algebraic      Sequence
 1               1      1
 2               1      1
 3               2      2
```

Advanced SPP Concepts

- *Implemented as a preprocessor*
- *Usage **similar** to Unix/C (e.g., STDIO)*
- *Call by reference, not value*
- *Executables may contain multiple tasks*
- *Subprocesses are cached by the CL*
- *Identifiers mapped to six characters (5+1)*

Advanced SPP Concepts (*cont.*)

- `define` *data structures with macros*
- `define` *inline functions with macros*
- *save these in* `include` *files*
- *Pointers are based on* `Mem_[]` *common*
- *Stack memory allocation (*`smark`/`salloc`/`sfree`*)*
- *Heap memory allocation (*`malloc`/`mfree`*)*
- *Catch errors with* `errchk` *and* `iferr` *blocks*
- *Generate errors with* `error` *and* `erract`

VOS Libraries

- *See the Quick Reference Card*
- *See the source code and system help docs*
- *Rich scientific and system APIs, see p. 41*
- *fmtio is similar to C stdio*
- `printf` *format specifications, see p. 51*
- *Intrinsic math functions, see p. 52*
- *Vector operators (VOPS), see p. 48*
- *VOPS also callable from IMFORT*

What’s Next?

Visit http://iraf.noao.edu

Send email to iraf@noao.edu