An SPP/VOS Quick Reference Card

The short name for each interface is in parentheses following the title. The IRAF directory for the source code is the same as the short name. Pertinent system header files (located in *lib$* or *hlib$*) are listed at the end of the title line. The names given to the arguments are meant to reflect their usage (and thus their datatype), but you may still find that you need to consult the source code. Some routines can be located through the help database: `help error opt=so`. Others can be listed directly: `page etc$pagefiles.x`. Examples are scattered throughout IRAF and usage is described more fully in §6 of *An Introductory User's Guide to IRAF SPP Programming*.

Command Language Input/Output (*clio*)

	`clgstr (param, outstr, maxch)`	**Get a string parameter.**
	`clpstr (param, string)`	**Output a string parameter.**
value =	`clget_ (param)`	**Get a typed parameter.**
	`clput_ (param, value)`	**Output a typed parameter.**
stat =	`clgcur (param, x,y,wcs, key,cmd,maxch)`	**Read a cursor parameter.**
stat =	`clgwrd (param, keyword, maxch, dict)`	**Look up a parameter in a dictionary.**
list =	`clpopni (param)`	**Open a file template or the `STDIN`.**
list =	`clpopns (param)`	**Open a sorted template.**
list =	`clpopnu (param)`	**Open an unsorted template.**
	`clpcls (list)`	**Close a file template.**
nfiles =	`clplen (list)`	**Return the number of files in a list.**
stat =	`clgfil (list, outstr, maxch)`	**Get the next file name.**
	`clcmdw (command)`	**Send a command to the CL and wait.**

File Input/Output (*fio*)

`<fset.h>`

fd =	`open (fname, mode, type)`	**Open a file for I/O.**
	`close (fd)`	**Close a file when finished.**
stat =	`read (fd, buffer, maxch)`	**Binary byte stream input.**
	`write (fd, buffer, maxch)`	**Binary byte stream output.**
	`flush (fd)`	**Flush the buffers immediately.**
stat =	`access (fname, mode, type)`	**Can the file be accessed?**
stat =	`fstati (fd, param)`	**Get the status of an open file.**
	`fseti (fd, param, value)`	**Set a FIO option.**
	`delete (fname)`	**Delete the named file.**
	`rename (old_fname, new_fname)`	**Rename a file.**
	`mktemp (root, fname, maxch)`	**Make a temporary file name .**
stat =	`protect (fname, action)`	**Protect or unprotect a file.**
stat =	`getline (fd, linebuf)`	**Get a line of text from a file.**
	`putline (fd, linebuf)`	**Output a line of text to a file.**

Image Input/Output (*imio*)

`<imhdr.h>, <imset.h>`

im =	`immap (image, mode, hdr_arg)`	**Map ("open") an image.**
	`imunmap (im)`	**Unmap ("close") an image.**
	`imflush (im)`	**Flush the image buffers.**
buf =	`imgl`*`N`*`_ (im [, line [, band]])`	**Get a line from an image, `N`=`[123]`.**
	`impl`*`N`*`_ (im [,line [,band]])`	**Output a line to an image, `N`=`[123]`.**
buf =	`imgs`*`N`*`_ (im, x1,x2 [,y1,y2 [,z1,z2]])`	**Get a section from an image, `N`=`[123]`.**
	`imps`*`N`*`_ (im, x1,x2 [,y1,y2 [,z1,z2]])`	**Output a section to an image, `N`=`[123]`.**
stat =	`imgnl_ (im, bufptr, v)`	**Generalized get next line.**
	`impnl_ (im, bufptr, v)`	**Generalized output next line.**
stat =	`imaccf (im, key)`	**Does the named keyword exist?**
	`imaddf (im, key, type)`	**Add, but don't initialize, a keyword.**
	`imadd_ (im, key, value)`	**Add or modify a header keyword.**
	`imastr (im, key, value)`	**Add or modify a string keyword.**
	`imdelf (im, key)`	**Delete a header keyword.**
value =	`imget_ (im, key)`	**Get a keyword of the specified type.**
	`imgstr (im, key, outstr, maxch)`	**Get a string valued keyword.**
	`imput_ (im, key, value)`	**Modify an existing header keyword.**
	`impstr (im, key, value)`	**Modify an existing string keyword.**
code =	`imgftype (im, key)`	**What is the keyword's datatype?**
	`imdelete (image)`	**Delete an (unmapped) image.**
imt =	`imtopenp (param)`	**Open an image template.**
	`imtclose (imt)`	**Close an image template.**
stat =	`imtgetim (imt, outstr, maxch)`	**Get the next image name.**
stat =	`imtrgetim (imt, index, outstr, maxch)`	**Get a randomly indexed image name.**
nimages =	`imtlen (imt)`	**Return the number of images in a list.**
	`imtrew (imt)`	**Rewind an image list.**

Memory Management (*memio*)

	`smark (sp)`	**Save the current stack pointer.**
	`salloc (ptr, nelem, type)`	**Allocate space on the stack.**
	`sfree (sp)`	**Pop the stack.**
	`malloc (ptr, nelem, type)`	**Allocate space on the heap.**
	`calloc (ptr, nelem, type)`	**Allocate and zero space.**
	`realloc (ptr, nelem, type)`	**Adjust the size of a buffer.**
	`mfree (ptr, type)`	**Free space on the heap.**

Graphics Input/Output (*gio*)

`<gset.h>`

gp =	`gopen (device, mode, fd)`	**Open a graphics stream.**
	`gclose (gp)`	**Close a graphics stream.**
	`gflush (gp)`	**Flush the graphics output.**
	`gline (gp, x1, y1, x2, y2)`	**Draw a line from `(x1,y1)` to `(x2,y2)`.**
	`gpline (gp, xa, ya, npts)`	**Draw a polyline.**
	`gmark (gp, x, y, type, xs, ys)`	**Draw a marker of a given size.**
	`gpmark (gp, xa, ya, npts, type, xs, ys)`	**Draw a polymarker.**
	`gamove (gp, x, y)`	**Move the pen to the absolute position.**
	`gadraw (gp, x, y)`	**Draw (absolute) from the current position.**
	`gseti (gp, param, value)`	**Set a GIO option.**
	`gswind (gp, x1, x2, y1, y2)`	**Set the window in the world coordinates.**
	`gsview (gp, x1, x2, y1, y2)`	**Set the viewport in normalized device coordinates.**
	`gascale (gp, array, npts, axis)`	**Scale the axis to fit the data.**
	`glabax (gp, title, xlabel, ylabel)`	**Draw and label the axes.**
	`gpagefile (gp, file, prompt)`	**Page a file from graphics cursor mode.**

Vector Operators (*vops*)

	`amov_ (a, b, npix)`	**Copy a vector.**
	`amovk_ (k, b, npix)`	**Copy a constant into a vector.**
	`aabs_ (a, b, npix)`	**Absolute value of a vector.**
	`aadd_ (a, b, c, npix)`	**Vector `c` is the sum of vectors `a` and `b`.**
	`aaddk (a, b, c, npix)`	**Vector `c` is the sum of vector `a` and scalar `b`.**
	`aavg_ (a, npix, mean, sigma)`	**Mean and sigma of a vector.**
value =	`amed_ (a, npix)`	**Return the median of a vector.**
	`abav_ (a, b, nblocks, blocksize)`	**Block average of a vector.**
	`abeq_ (a, b, c, npix)`	**`c[i]=1` if `a[i] == b[i]`, else `c[i]=0`.**
	`alim_ (a, npix, minval, maxval)`	**Compute the min and max of a vector.**

Miscellaneous (*etc*)

	`error (code, message)`	**Generate an error action (may be trapped).**
	`erract (severity)`	**Take an error action.**
len =	`envgets (key, value, maxch)`	**Fetch the string value of an environment variable.**
value =	`envget_ (key)`	**Return the typed value of an environment variable.**
	`qsort (array, nelems, compare)`	**Sort an integer array by the function `compare ()`.**
	`qqsort (array, nelems, compare, arg)`	**Sort by a function with an argument.**
	`pagefiles (files)`	**Page text file(s) on the STDOUT.**
	`pagefile (file, prompt)`	**Page a text file on the STDOUT.**
real =	`urand (seed)`	**Uniform "random" number in the interval (0,1).**
	`sysid (outstr, maxch)`	**Return a system and user identification string.**
int =	`btoi (bool)`	**Convert a boolean to an integer (`YES` or `NO`).**

Formatted Input/Output (*fmtio*)

<chars.h>, <ctype.h>

	printf (format)	**Begin a print to the** STDOUT**.**
	eprintf (format)	**Begin a print to the** STDERR**.**
	fprintf (fd, format)	**Begin a print to a file.**
	sprintf (outstr, maxch, format)	**Begin a print to a string.**
	clprintf (param, format)	**Begin a print to a CL parameter.**
	parg_ (value)	**Complete a typed format.**
	pargstr (string)	**Complete a string format.**
stat =	scan ()	**Begin a scan from the** STDIN**.**
stat =	fscan (fd)	**Begin a scan from file.**
stat =	sscan (str)	**Begin a scan from a string.**
stat =	clscan (param)	**Begin a scan from a CL parameter.**
	garg_ (value)	**Get a typed value.**
	gargstr (outstr, maxch)	**Get the rest of the line.**
	gargwrd (outstr, maxch)	**Get the next "word".**
stat =	strdic (input, keyword, maxch, dict)	**Look up a string in a dictionary.**
bool =	streq (string1, string2)	**Compare two strings for equality.**
	strcpy (string1, string2, maxch)	**Copy** string1 **to** string2**.**

Format Specifications

An SPP format specification has the form "*%w.dC*", where *w* is the field width, *d* is the number of decimal places or the number of digits of precision, and *C* is the format code. The *w* and *d* fields are optional. The format codes *C* are as follows:

b	**boolean (**YES **or** NO**)**
c	**single character (**c**,** \c**, or** \nnn**)**
d	**decimal integer**
e	**exponential format,** *d* **is the precision**
f	**fixed format,** *d* **is the number of decimal places**
g	**general format,** *d* **is the precision**
h	*hms* **format (**hh:mm:ss.ss**,** *d* **is the number of decimal places)**
H	*HMS* **format, convert from degrees to hours first (divide by 15)**
m	*ms* **or** *hs* **format (**mm:ss.ss**),** *d* **is the number of decimal places**
M	*MS* **or** *HS* **format, convert from degrees to hours first**
o	**octal integer**
r*N*	**convert integer to or from radix** *N*
s	**string,** *d* **is the maximum number of chars to print**
t	**advance to column given by** *w*
u	**unsigned decimal integer**
w	**output the number of spaces given by** *w*
x	**hexadecimal integer**
z	**complex format (**(r,r)**),** *d* **is the precision**
*	**deferred, get the field from the next** parg_ **call**

Conventions for specifying the field width:

w = n	**right justify and blank fill in a field of** n **characters**
w = -n	**left justify and blank fill in a field of** n **characters**
w = 0n	**right justify and zero fill in a field of** n **characters**
w = 0	**use as much space as needed**
absent	**same as** *w* = 0

Escape sequences for string literals and character constants:

\b	**backspace**
\f	**form feed**
\n	**newline (**<CR><LF>**)**
\r	**carriage return**
\t	**tab**
\"	**string delimiter character**
\'	**character constant delimiter character**
\\	**backslash character**
\nnn	**octal value of character**

Any combination of the fields *w*, *d*, *C*, or *N* may be specified as asterisks (*) in the format string, allowing the field to be passed at run time in a parg_ call.

SPP Intrinsic Functions

Type Conversions

char = char (integer)	**Convert to character.**
short = short (z)	**Convert to short.**
int = int (z)	**Truncate to integer.**
int = nint (x)	**Round to integer.**
long = long (z)	**Convert to long integer.**
real = real (z)	**Convert to real.**
real = aimag (complex)	**Imaginary part.**
double = double (z)	**Convert to double precision.**
complex = complex (z)	**Convert to complex.**

Trigonometry (angles are in radians)

y = sin (y)	**Sine.**
y = cos (y)	**Cosine.**
x = tan (x)	**Tangent.**
x = asin (x)	**Inverse sine.**
x = acos (x)	**Inverse cosine.**
x = atan (x)	**Inverse tangent.**
x = atan2 (x1, x2)	**Inverse tangent of** x1/x2**.**
x = sinh (x)	**Hyperbolic sine.**
x = cosh (x)	**Hyperbolic cosine.**
x = tanh (x)	**Hyperbolic tangent.**

Miscellaneous

w = abs (z)	**Absolute value.**
complex = conjug (complex)	**Complex conjugate.**
w = min (w1, w2, ...)	**Minimum value.**
w = max (w1, w2, ...)	**Maximum value.**
w = mod (w1, w2)	**Remainder after** w1/w2**.**
y = sqrt (y)	**Square root.**
y = log (y)	**Natural logarithm.**
x = log10 (x)	**Common logarithm.**
y = exp (y)	**Exponentiation.**

The allowed datatypes of the arguments and returned values are:

```
w = integer, real, or double
x =          real or double
y =          real,   double, or complex
z = integer, real,   double, or complex
```

Do not use short or long integer arguments with any functions other than the type conversion intrinsics. The datatypes must match for functions with more than one argument. The datatype returned by the functions is the same as the arguments, except for type conversions or the absolute value of a complex number. SPP performs the "normal" automatic type conversions in expressions.

National Optical Astronomy Observatories
P.O. Box 26732, Tucson, AZ 85726-6732