URL: http://cxc.harvard.edu/ciao3.4/paramio.html
Last modified: December 2006

AHELP for CIAO 3.4

paramio

Context: modules

Jump to: Description Example HANDLING PARAMETERS IN A SCRIPT CHANGES IN CIAO 3.2 See Also

Synopsis

The S−Lang interface to the CXC parameter system

Description

The paramio module is the interface between the S−Lang interpreter (see "ahelp slang") and the CXC parameter library (see "ahelp parameter"). This document provides an overview of the features of the paramio module, and tips for using it efficiently in a S−Lang program. Detailed descriptions of each function are provided by individual ahelp pages.

The paramio library is not available by default; to use it in a S−Lang program, it must be loaded using the S−Lang require() function:

```
require("paramio");
```

Functions provided by the module

The following functions are provided by the module (also try "ahelp −c paramio"); use "ahelp <function>" to get a detailed description of a function:

Function name with arguments
paramopen(paramfile)
paramopen(paramfile, mode)
paramopen(paramfile, mode, paramlist)
paramclose(paramfile)
pset(paramfile, parname, value)
pset(paramfile, pars_assoc_array)
pget(paramfile, param)
pquery(paramfile, param)
paccess(paramfile)
paccess(paramfile, mode)
paramgetpath(paramfile)

punlearn(paramfilespec)
plist_names(paramfile)
plist_mode(paramfile, mode)
plist_type(paramfile, type)
pgets(paramfile, param)
pputs(paramfile, param, shortval)
pgeti(paramfile, param)
pputi(paramfile, param, intval)
pgetb(paramfile, param)
pputb(paramfile, param, intval)
pgetf(paramfile, param)
pputf(paramfile, param, floatval)
pgetd(paramfile, param)
pputd(paramfile, param, doubleval)
pgetstr(paramfile, param)
pputstr(paramfile, param, stringval)

The module also defines a variable − PF_Errno − and a number of constants, for defining a mode, datatype, or error status (the error values are integers whilst the mode and datatype values are strings).

Error	Mode	Datatype
PF_MALLOC_ERROR	PF_MODE_AUTO	PF_BOOL
PF_NUMBER_FORMAT_BAD	PF_MODE_QUERY	PF_INT
PF_CORRUPT_FIELD	PF_MODE_LEARN	PF_REAL
PF_NOT_IMPLEMENTED	PF_MODE_HIDDEN	PF_STRING
PF_RANGE_ERROR		PF_FILETYPE
PF_UNKNOWN_PARAMETER		PF_PSET
PF_ACCESS_ERROR		
PF_FILE_NOT_FOUND		
PF_FILE_OPEN_ERROR		
PF_BAD_ARGUMENT		
PF_IO_ERROR		
PF_INDIRECT_ERROR		
PF_UNKNOWN_ERROR		

Accessing parameter files

Except for paramopen() and paramclose(), all the functions accept either a Param_File_Type or a name to denote the parameter file to operate upon. The use of filenames is more convenient for simple operations, but results in an implicit file open/close per call. For more iterative/intensive I/O operations, better performance will result from the use of the file pointer returned by paramopen().

Example

```
chips> require("paramio")
chips> punlearn("dmextract")
chips> pget("dmextract","opt")
pha1
chips> pset("dmextract","infile","in.fits")
chips> pget("dmextract","infile")
in.fits
chips> print( plist_mode("dmextract",PF_MODE_AUTO) )
infile
outfile
chips> print( plist_type("dmextract",PF_REAL) )
bkgnorm
sys_err
```

HANDLING PARAMETERS IN A SCRIPT

The following example routine can be used in S−Lang scripts to simplify the parameter handling. It is a simplified version of the routine used in the $ASCDS_INSTALL/contrib/bin/acis_fef_lookup S−Lang script, which can be examined for further details. To keep the example short much of the error checking used in the acis_fef_example script has been removed.

```
%
% Usage:
%  ( paramlist, progname ) = process_params( argv );
%
require("paramio");
define process_params ( argv ) {

  % can we find the parameter file?
  variable progname = path_basename(argv[0]);
  if ( progname == NULL ) return ( NULL, NULL );

  % open the parameter file
  %
  variable mode = "rw";
  variable fp = paramopen( NULL, mode, argv );
  if ( fp == NULL ) return ( NULL, progname );

  % what are the parameters as a function of type?
  variable ptypes = Assoc_Type [String_Type];
  variable ptype, pname;
  foreach ( [ PF_BOOL, PF_INT, PF_REAL, PF_STRING, PF_FILENAME ] ) {
    ptype = ();
    foreach ( plist_type( fp, ptype ) ) {
      pname = ();
      ptypes[pname] = ptype;
    }
  }

  % The call to pquery returns the specified parameter value (and
  % queries the user to find it out if necessary)
  variable params = Assoc_Type [];
  foreach ( plist_names( fp ) ) {
    pname = ();
    ptype = ptypes[pname];
    variable pstr  = pquery( fp, pname );
```

```
      switch ( ptype )
      { case PF_STRING:   params[pname] = pstr; }
      { case PF_FILENAME: params[pname] = pstr; }
      { case PF_REAL:     params[pname] = atof(pstr); }
      { case PF_INT:      params[pname] = integer(pstr); }
      { case PF_BOOL:     params[pname] = pstr; }
    }

    % close the parameter file
    paramclose( fp );

    % return the associative array of parameters and the program name
    return ( params, progname);

  } % process_params()
```

With the above function definition, one can parse arguments sent in to a S−Lang script using (again neglecting the error checks):

```
  % process the parameters
  variable plist, progname;
  ( plist, progname ) = process_params( __argv );

  % print out the parameter values
  vmessage( "Parameters for %s are:", pname );
  message( "\tName\tType\t\tValue" );
  foreach ( plist ) using ( "keys", "values" ) {
    variable k, v;
    ( k, v ) = ();
    vmessage( "\t%s\t%s\t%s", k, string(typeof(v)), string(v) );
  }
```

The __argv array is created automatically by slsh and contains a list of the arguments specified on the command line (with the first element being the name of the program executed).

Since, in most scripts, you know the names of the parameters so you can access the elements directly. For instance, if the parameter file had parametes "infile", "outfile", and "verbose", you could say:

```
  variable plist;
  ( plist, ) = process_params( __argv );
  variable infile  = plist["infile"];
  variable outfile = plist["outfile"];
  variable verbose = plist["verbose"];
```

CHANGES IN CIAO 3.2

The module can now be loaded by using the

```
require("paramio");
```

statement, although the previous method (loading with the import command) still works.

The paccess() routine now has two modes of operation, determined by the type of the first variable in its argument list. See "ahelp paramio paccess" for further information.

See Also

calibration

caldb

chandra

coords, guide, isis, level, pileup, times

chips

chips

concept

autoname, parameter, stack, subspace

dm

dm, dmbinning, dmcols, dmfiltering, dmimages, dmimfiltering, dmintro, dmopt, dmregions, dmsyntax

gui

gui

modules

pixlib, stackio

paramio

paccess, paramclose, paramopen, pget, pgets, plist_names, pquery, pset, punlearn

slang

overview, slang, tips

tools

dmhistory, dmkeypar, dmmakepar, dmreadpar, paccess, pdump, pget, pline, plist, pquery, pset, punlearn

---

URL: http://cxc.harvard.edu/ciao3.4/paramio.html
Last modified: December 2006