INTEGRAL Science Data Centre

Parameter Interface Library Users Manual

Reference	:	ISDC/PIL
Issue	:	1.8.5
Date	:	19 SEP 2002

INTEGRAL Science Data Centre
Chemin d'Écogia 16
CH–1290 Versoix
Switzerland

http://isdc.unige.ch

Authors and Approvals

ISDC	Parameter Interface Library Users Manual	
19 SEP 2002	1.8.5	PIL 1.8.5

Prepared by : J. Borkowski

QA checked by : T. Lock ....

Approved by : R. Walter ....

Document Status Sheet

ISDC	Parameter Interface Library Users Manual	
31 AUG 1998	1.0	PIL Users Manual released
19 MAR 1999	1.2	PIL Users Manual released
25 JUN 1999	1.3	PIL Users Manual released
12 FEB 2001	1.4	PIL Users Manual released
10 MAR 2002	1.8	PIL Users Manual released
06 MAY 2002	1.8.2	PIL Users Manual released
19 SEP 2002	1.8.5	PIL Users Manual released
19 SEP 2002	Printed	

Contents

List of Reference Documents . . . v

Glossary of Terms . . . vi

I Getting started **2**

1 Building the PIL library . . . 3

1.1 Necessary tools . . . 3

1.2 Unpacking distribution file and setting up directories structure . . . 4

1.3 Autoconfiguring Makefiles . . . 4

1.4 Compilation of library and demo programs . . . 5

1.5 Installation of library . . . 5

1.6 Sample programs . . . 5

2 IRAF parameter files . . . 6

3 How parameter files are named and where are they looked for ? . . . 7

4 PIL extensions to IRAF parameter files format . . . 8

4.1 Parameter file line length . . . 8

4.2 Expansion of environment variables . . . 8

4.3 Vector support . . . 8

5 How parameters are evaluated . . . 9

5.1 Positional parameters . . . 10

5.2 Named parameters . . . 10

II PIL C/C++ Language Interface **11**

6 PIL C/C++ Language API . . . 12

6.1 Introduction . . . 12

6.2 PIL C/C++ include files . . . 12

6.3 C/C++ API functions . . . 12

7 Calling PIL library functions from C/C++ . . . 28

III PIL Fortran 90 Language Interface **31**

8 PIL F90 Language API . . . 32

8.1 Introduction . . . 32

8.2 PIL Fortran 90 module files 32

8.3 Fortran 90 API functions 32

9 Calling PIL library functions from Fortran 90 46

IV Appendices **49**

A PIL Error Codes 50

List of Reference Documents

[ESA/PSS-05-0]	ESA Software Engineering Standards	Issue 2.0
[ISDC/CTS]	ISDC Coding and Testing Standards	Issue 2.1
[ISDC/SCMP]	ISDC Software Configuration Management Plan	Issue -
[ISDC/SQAP]	ISDC Software Quality Assurance Plan	Issue -
[ISDC/SVVP]	ISDC Software Verification and Validation Plan	Issue -

Glossary of Terms

- *Analysis software:* Software written for the purpose of analyzing data.
- *Component:* A well defined element of software within the ISDC system that performs a specific task – can be an executable or library.
- *Delivery Module:* A set of properly organized source files making up a software component that are given to ISDC for integration into the overall system.
- *Development team:* One of the teams (e.g., ISDC, SPI, IBIS, Jem-X, OMC, others) writing software for the ISDC system.
- *Driver:* Software which 'drives' the execution of a *unit* under test in order to provide a framework for setting input parameters, to control and monitor the execution, and to report the test results.
- *Exported header file:* A source code header file intended for use by programs outside the software component in which the header file resides.
- *Identifiers:* The names given to functions, subroutines, and variables within source code.
- *Internal header file:* A source code header file intended for use only within the software component in which the header file resides.
- *Parameter Files:* The elements (files) of a component that define the inputs needed from the system in order for the component to perform its function.
- *Pipeline:* A set of programs grouped together with flow control instructions (e.g., tests and loops). When a pipeline is executed, the programs involved run automatically in a specific order
- *Reference Platform:* The configuration of hardware and operating system software on which all other software is expected to execute.
- *System Release:* A fully tested and complete instance of the ISDC system that has been deemed ready for use in operations.
- *System software:* Software written for the purpose of implementing the ISDC system infrastructure.

Abstract

The Parameter Interface Library (PIL) is an ANSI C/C++ and Fortran 90 callable library which manages access to IRAF/XPI compatible parameter files. In general each program (executable) written for ISDC will have its parameter file. PIL gives standard set of functions (API) which can be called by applications wanting to access parameter files. PIL functions allow for : reading/writing individual parameters from parameter file, automatic selection of server mode, (re)opening/closing of parameter file, querying information about parameter file. C/C++ bindings have (almost) one to one corresponding Fortran bindings. Minor differences are result of different calling conventions in C/C++ and Fortran 90.

Version 1.6.x of PIL adds support for enumerated values, fixed and variable length vectors of values, and expansion of environment variables specified in parameters.

This document describes version 1.8.5 of PIL

Part I

Getting started

This chapter explains how to build and install the PIL library from its source code distribution. It also provides some initial help on programming with PIL by reviewing the sample programs that come with the PIL distribution.

1 Building the PIL library

1.1 Necessary tools

1.1.1 Hardware

Version 1.8.2 of PIL (May 2002) compiles on the following architectures :

```
sparc-sun-solaris    (32 / 64 bit)
intel-gnu-linux      (32 bit)
```

The library may compile/run on other architectures (we have reports of successful compilation on HP-UX, mips-sgi-irix and alpha-dec-osf machines), but PIL development team has access only to those aforementioned. Furthermore ISDC officially supports only "reference platform", which is defined as sparc-sun-solaris running Solaris 8 with Forte 6 compiler set. Refer to :

```
http://isdc.unige.ch/index.cgi?Software+swplatform
```

for more information.

1.1.2 Software

Version 1.8.2 of PIL was tested with following compilers :

```
on Solaris 8 :

  Sunsoft cc C compiler version 5.0, 5.1, 6.x (aka Forte 6)
  Sunsoft CC C++ compiler version 5.0, 5.1, 6.x (aka Forte 6)
  Sunsoft f90 compiler version 5.0, 5.1, 6.x (aka Forte 6)

on Linux kernel version 2.2/2.4 :

  gcc/g++ C/C++ compiler version 2.7.*, 2.8.*, 2.95.2, 3.0.*
  Fujitsu Fortran 95 Express Version 1.0 (now Lahey ver 5+)
```

PIL library may be compilable with other compilers, however the PIL development team have access to only those aforementioned. Not all combinations of C/C++ and F90 compilers were tested.

Active development and debugging is done on Solaris with some work also on Linux.

Besides compilers, one needs the following system utilities :

```
gnu make (ver. 3.79.1) (Sun's /usr/ccs/bin/make does _NOT_ work)
gzip
tar (or gtar but renamed to tar)
```

1.2 Unpacking distribution file and setting up directories structure

PIL library is usually delivered as a part of support-sw package. It is however possible to compile it as a standalone package.

After downloading the distribution file, one has to move it to empty directory in which she/he has read and write access rights. Then the following should be done :

```
gzip -d pil-1.8.2.tar.gz
tar xvf pil-1.8.2.tar
```

First line uncompresses the distribution file, the second unpacks files from tar-file, usually creating several files and subdirectories. Main directory should contain among the others Makefile.in, makeisdc1.in, configure.in, configure files and autoconf subdirectory.

1.3 Autoconfiguring Makefiles

Before attempting autoconfiguration the directory tree has to be set to the clean state. This is done with the following command :

```
make distclean
```

This command deletes any existing object files, executables produced by build process, config.cache, Makefiles produced by previous run of configure script.

Before running configure one may wish to review (and probably manually edit) files makeisdc1.in and Makefile.in. makeisdc1 and Makefile should not be edited since their contents will be overwritten by subsequent run of $ISDC_ENV/bin/ac_stuff/configure script.

To autoconfigure PIL library one has to run :

```
$ISDC_ENV/bin/ac_stuff/configure
```

$ISDC_ENV/bin/ac_stuff/configure script adapts Makefile to C and F90 compilers tastes. If there is no F90 compiler available configure script disables compilation of F90 source files. If configure script is unable to automatically find correct C and F90 compiler one can set CC, F90, CFLAGS and F90FLAGS environment variables to force configure script to choose specific compilers/options. For example configure script tends to favor gcc over other C compilers. So if you have gcc and other C compiler it will always choose gcc. If you want to use C compiler other then gcc please type :

```
setenv CC name_of_your_C_compiler (cc is quite common)
```

before running ./configure

Notes

On some architectures NAG F90 compiler requires -Qpath option to be predefined in F90FLAGS. configure script is not smart enough to figure out where NAG F90's libraries/executables reside (usually not in PATH). Besides, NAG F90 compiler is not supported.
If CFLAGS and F90FLAGS setup by ./configure script are incorrect one may fix them directly in Makefile/makeisdc1. For instance, some people want to have libraries compiled with debug option turned on ("-g"), and configure script does not always put "-g" option in CFLAGS/F90FLAGS. Please keep in mind that any subsequent run of configure script will overwrite any changes.

1.4 Compilation of library and demo programs

To compile PIL library and build demo executables one has to type :

```
make
```

This will create libpil.a file and executables :pset, plist, pilcdemo, pilfdemo among others.

1.5 Installation of library

To install PIL package under $ISDC_ENV directory tree one has to type :

```
make global_install
```

For more information about make files and installation procedure refer to makefiles-2.4.4 manual (found on ISDC web serwer) or README.make file.

1.6 Sample programs

A small number of sample programs is distributed together with PIL library (and built when compiling library). They are :

- pset - PIL's equivalent for IRAF/XPI pset program. Sets value of parameter in arbitrary parameter file.
- pget - PIL's equivalent for IRAF/XPI pget program. Read value of parameter in arbitrary parameter file and display it on screen.
- plist - PIL's equivalent for IRAF/XPI plist program. Dumps contents of parameter file on screen.
- pil_lint - program checks given parameter file and reports any errors encountered.
- pil_gen_c_code - given parameter file program dumps (to stdout) source of C program which can read that parameter file.
- pilcdemo - sample C application (see source code for more info)
- pilfdemo - sample F90 application (see source code for more info)
- cvector - test program for vector support
- ISDCcopy - sample CTS compliant application (use make ISDCcopy to build it - requires support-sw installed)

Most of those demo program are for testing purposes, and do not perform any useful calculations. Refer to chapters 7 and 9 for information how to use PIL library.

2 IRAF parameter files

The purpose of PIL library is to enable ISDC applications access IRAF compatible parameter files. IRAF parameter file is a text file. Every line describes single parameter. Format is as follows :

```
name,type,mode,default,min,max,prompt
```

- `name` : name of parameter
- `type` : type of parameter. Allowable values:
  - `b` : means parameter of boolean type
  - `i` : means parameter of integer type
  - `r` : means parameter of real type
  - `s` : means parameter of string type
  - `f` : for parameter of filename type. `f` may be followed by any combination of r/w/e/n meaning test for read access, write access, presence of file, absence of file. Thus fw means test whether file given as a value of parameter is writable.
- `mode` : mode of parameter. Allowable value is any reasonable (it is: a/h/q/hl/ql) combination of :
  - `a/auto` : effective mode equals to the *value* of parameter named `mode` in parameter file. If that parameter is not found in parameter file or is found invalid then the effective mode is 'hidden'.
  - `h/hidden` : No questions are asked, unless default value is invalid for given data type. for example "qwerty" is specified as a value for integer parameter. Note that is PILOverrideQueryMode function was called with argument set to PIL_QUERY_OVERRIDE then no questions are asked even for parameters with invalid values.
  - `l/learn` : If application changed parameter value, new value will be written to parameter file when application terminates. Actually this takes places when application calls either PILClose() or PILFlushParameters() (or CommonExit() when writing ISDC's CTS compliant software). If this flag is not specified then any changes to the parameter (via PILGetXXX or PILPutXXX) are lost and are not written to disk.
  - `q/query` : Always ask for parameter. The format of the prompt is : prompt field from parameter file (parameter name if prompt fielf is empty) followed by allowable range (if any) in ¡¿, followed by default value (if any) in [], followed by colon. Pressing RETURN key alone accepts default value. If newly entered value (or default value in the case RETURN key alone is pressed) is unacceptable library prompts user to reenter value. If the value is overridden by command line argument, then the PIL library does not prompt for that parameter (if value is valid and within boudaries if any) Note that is PILOverrideQueryMode function was called with argument set to PIL_QUERY_OVERRIDE then no questions are asked ever (even for parameters with invalid values).
- `default` : default value for parameter [this field is optional]. This can be: yes/no/y/n for booleans, integer/real literals (ie. 123, -34567 for integers, 1.23, 1234, -45.3e-5 for reals) for integers/reals, and string literals for strings and filenames. String literals can be: abcdef, "abcdef", 'abcdef'.
- `min` : minimum value allowable for parameter [this field is optional]. Range checking is enabled only if both `min` and `max` fields are nonempty. See also `max`. When `max` field is empty then `min` field can also contain pipe symbol (vertical bar) separated list of allowable values for given parameter. This works for integer, real, string and filename types. For instance :

```
OutFile,f,ql,"copy.o",/dev/null|copy.o|dest.o,,"Enter output file name"
```

specifies that OutFile parameter can take only 3 distinct values, it is /dev/null, copy.o or dest.o. Furthermore, for string (and _ONLY_ for string, this does _NOT_ apply for filename parameter types) parameter types, case does not matter, and string returned by PILGetString is converted to uppercase. Note, that automatic conversion to uppercase is only done when enum list is specified for given parameter.

- `max` : maximum value allowable for parameter [this field is optional]. Works for integer, real string and filename data types. Both `min` and `max` must be specified for range checking to be active. Also `min` cannot be larger than `max`. In case of any format error in `min` or `max` PIL assumes that no range checking is requested. See also `min` for a description of enumerated value list.
- `prompt` : short description of parameter to be displayed whenever library asks for value. If none is given library will display parameters name.

As an example :

```
pressure,r,ql,1013,,,"Enter atmospheric pressure in hPa"
```

describes parameter named pressure of type real, mode = query + learn, with default value = 1013, no range checking and prompt

```
"Enter atmospheric pressure in hPa"
```

Empty lines and lines beginning with '#' are considered to be comment lines.

3 How parameter files are named and where are they looked for ?

Typically for every executable there is a corresponding parameter file. The name of parameter file is the name of executable file plus ".par" suffix. Thus executable :

```
isdc_copy
```

will have parameter file named :

```
isdc_copy.par
```

Parameter files are searched for in several locations:

- PFILES enviromnent variable

  This variable is used to specify where the parameters are looked for. The variable uses a ";" delimiter to separate 2 types of parameter directories:

  ```
  <path1>;<path2>
  ```

where path 1 is one or more user/writeable parameter directories, and path 2 is one or more system/read-only parameter directories. When path 1 equals path 2 one can omit path 2 and semicolon. The PIL library allows multiple ":"-delimited directories in both portions of the PFILES variable. The default values from path 2 are used the first time task is run, or whenever the default values have been updated more recently than the user's copy of the parameters. The user's copy is created when a task terminates, and retains any learned changes to the parameters.

- Current working directory.

  It is equivalent of PFILES set to ".".

Notes

contrary to FTOOLS/SAOrd/XPI PFILES environment variable can be left undefined. In this can case PIL library will look for parameter files only in current directory. FTOOLS executables return error in this situation.

4 PIL extensions to IRAF parameter files format

This section lists PIL extensions to IRAF parameter files.

4.1 Parameter file line length

PIL limits parameter file length to 2000 characters. This is much larger that IRAF parameter file line length limit (80 or 255 depending on application/library). Thus parameter files created by PIL may not necessarily be readable by IRAF.

4.2 Expansion of environment variables

If value of given parameter contains the following string :

```
${ANY_VAR_NAME}
```

PIL expands it to the value of ANY_VAR_NAME environment variable. Any number of environment variables can be specified. Also the same variable can be specified many times. As an example, for a user joe with home directory in /home/joe :

```
${HOME}/pfiles/${HOME}
```

is expanded to

```
/home/joe/pfiles/home/joe
```

4.3 Vector support

PIL internally supports parameters with values with specify vectors with either fixed or variable number of elements. This works for parameter types: integer, real and real4. The parameter itself has to be of type string (type stored in parameter file). Thus it can be read as either string (by calling PILGetString) or vector of values (by calling PILGetxxxVector or PILGetxxxVarVector). See description of PILGetxxxVector or PILGetxxxVarVector functions for more details.

5 How parameters are evaluated

In general parameter's values are read from the parameter file. They can however be overridden if the command line arguments are entered. Let's assume that we have written simple application named ISDCCopy to copy a file. It accepts 2 parameters, namely InFile and OutFile. The corresponding parameter file could be :

```
InFile,s,ql,sample1.fits,,,"Enter input file name"
OutFile,s,ql,/dev/null,,,"Enter output file name"
```

If we run that application without any arguments

```
ISDCCopy
```

it will prompt us for these two parameters. Pressing 2 times **RETURN** key will accept default values and in this case application will attempt to copy sample1.fits file onto /dev/null device (not a very useful function). If we type :

```
ISDCCopy sample34.fits
```

or

```
ISDCCopy InFile="sample34.fits"
```

it will ask only for 2nd second parameter. The value of first parameter will be set to sample34.fits. If we type:

```
ISDCCopy sample45.fits mycopy.fits
```

or

```
ISDCCopy InFile="sample45.fits" OutFile="mycopy.fits"
```

or

```
ISDCCopy OutFile="mycopy.fits" InFile="sample45.fits"
```

it will not ask for any parameters. If we type :

```
ISDCCopy OutFile="mycopy.fits"
```

it will ask only for the first parameter.

Notes

If a call to PILOverrideQueryMode has been made and PIL_QUERY_OVERRIDE mode is in effect PIL library does not prompt user when invalid or out of range argument is encountered. Instead it returns with an error.

Bogus parameter names passed in command line (i.e. those without maching counterpart in parameter file) may result in application returning with an error PIL_BOGUS_CMDLINE. This can happen if application calls PILVerifyCmdline(). PILInit() itself does not call PILVerifyCmdline().

5.1 Positional parameters

When specifying new value for a parameter on the command line, the simplest method is to type its value only :

```
exename par1value par2value par2value [ ... ]
```

More specifically, PIL treats n-th command line argument as new value for n-th parameter in the parameter file (hence the name *positional*). That is, assuming there are no empty/comment lines, the parameter in the n-th line in the parameter file.

5.2 Named parameters

A more flexible method to specify new value of the parameter is to use the syntax :

```
name=value
```

Using this syntax, one can specify new value for the parameters in any order. Thus :

```
ISDCCopy OutFile="mycopy.fits" InFile="sample45.fits"
```

and

```
ISDCCopy InFile="sample45.fits" OutFile="mycopy.fits"
```

are equivalent. Furthermore, since spaces are allowed around '=' separator, than all the following formats are equivalent :

```
ISDCCopy OutFile=mycopy.fits
ISDCCopy OutFile =mycopy.fits
ISDCCopy OutFile= mycopy.fits
ISDCCopy OutFile = mycopy.fits
```

note: any positional parameters must precede parameters given with name=value syntax.

Part II

PIL C/C++ Language Interface

6 PIL C/C++ Language API

6.1 Introduction

This section describes the C language implementation of the Parameter Interface Library Application Programming Interfaces (PIL APIs) It also gives C/C++ languages specific information necessary to use those APIs. Users interested only in using PIL from Fortran 90 applications should read section 8.

PIL library is standalone and can be compiled independently of other ISDC libraries.

Unless stated otherwise all PIL API functions return status code of type int. This is either ISDC_OK which equals PIL_OK which equals 0, which means everything went perfectly or negative value (error code) meaning some error occured. List of error codes can be found in pil_error.h file. Functions given below are the "official" ones. Internally PIL library calls many more functions.

6.2 PIL C/C++ include files

Applications calling PIL services from C/C++ source code should include `pil.h`. Alternatively they can include `isdc.h` which includes `pil.h` file. `pil.h` file in turn internally includes all other PIL relevant include files.

`pil.h` file can be called from either C or C++ source code. It contains prototypes of all C/C++ API functions, definitions of constants, declarations of global variables and definitions of data structures.

6.3 C/C++ API functions

6.3.1 PILinit

```
int PILInit(int argc, char **argv);
```

Description
This function initializes PIL library. This function has to be called before any other PIL function (there are some exceptions to this rule). It does the following :
Based on PILModuleName (or argv[0] if PILModuleName is empty) calculates name of parameter file. Usually name of parameter file equals argv[0] + ".par" suffix but this can be overriden by calling PILSetModuleName and/or PILSetModuleVersion before calling PILInit. After successful termination parameter file is opened and read in, and global variable PILRunMode is set to one of the following values :

```
ISDC_SINGLE_MODE
ISDC_SERVER_MODE
```

ISDC_SERVER_MODE is set whenever there is parameter "ServerMode" and its value is "yes" or "y". In any other case PILRunMode is set to ISDC_SINGLE_MODE.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `int argc [In]`
  number of command line arguments
- `char **argv [In]`
  array of pointers to command line arguments. argv[0] is typically name of the executable.

Notes
Only one parameter file can be open at a time. Parameter file remains open until PILClose is called. When writing applications for ISDC one should not use PILInit directly. Instead, one should call CommonInit function (from Common library) which calls PILInit.

6.3.2 PILClose

```
int PILClose(int status);
```

Description
This function has to be called before application terminates. It closes open files, and writes all learned parameters to the disk files (only when status == ISDC_OK). Once this function is called one cannot call any other PIL functions. One can however call PILInit to reinitialize PIL library. Function also clears PILRunMode, PILModuleName and PILModuleVersion global variables.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `int status [In]`
  PIL_OK/ISDC_OK/0 means normal return, any other value means abnormal termination. In this case changes to parameters made during runtime are NOT written to parameter files.

Notes
This function does not terminate process. It simply shuts down PIL library. When writing applications for ISDC one should not use PILClose directly. Instead, one should call CommonExit function (from Common library) which calls PILClose.

6.3.3 PILReloadParameters

```
int PILReloadParameters(void);
```

Description
This function reloads parameters from parameter file. It is called internally by PILInit. It should be called explicitly by applications running in ISDC_SERVER_MODE to rescan parameter file and reload parameters from it. Current parameter list in memory (including any modifications) is deleted. PIL library locks whole file for exclusive access when reading from parameter file.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes
parameter file remains open until PILClose is called. Function internally DOES NOT close/reopen parameter file. Application should call PILFlushParameters before calling this function, otherwise all changes made to parameters so far are lost.

6.3.4 PILFlushParameters

```
int PILFlushParameters(void);
```

Description
This function flushes changes made to parameter list (in memory) to disk. Current contents of parameter file is overwritten. PIL library locks whole file for exclusive access when writing to parameter file.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes
parameter file remains open until PILClose is called.

6.3.5 PILSetModuleName

```
int PILSetModuleName(char *name);
```

Description
Sets name of the module which uses PIL services. Result is stored in global variable PILModuleName. Usually name of parameter file equals argv[0] + ".par" suffix but this can be overriden by calling PILSetModuleName and/or PILSetModuleVersion before calling PILInit.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  new module name

6.3.6 PILSetModuleVersion

```
int PILSetModuleVersion(char *version);
```

Description
Sets version of the module which uses PIL services. Result is stored in global variable PILModuleVersion.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *version [In]`
  new module version. If NULL pointer is passed version is set to "version unspecified" string

6.3.7 PILGetParFileName

```
int PILGetParFilename(char **fname);
```

Description
This function retrieves full path of used parameter file. Absolute path is returned only when PFILES environment variable contains absolute paths. If parameter file is taken from current dir then only filename is returned. Pointer returned points to a statically allocated buffer, applications should copy data from it using strcpy.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char **fname [Out]`
  pointer to name/path of the parameter file

6.3.8 PILOverrideQueryMode

```
int PILOverrideQueryMode(int newmode);
```

Description
This functions globally overrides query mode. When newmode passed is PIL_QUERY_OVERRIDE, prompting for new values of parameters is completely disabled. If value is bad or out of range PILGetXXX return immediately with error without asking user. No i/o in stdin/stdout is done by PIL in this mode. When newmode is PIL_QUERY_DEFAULT PIL reverts to default query mode.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `int newmode [In]`
  new value for query override mode

6.3.9 PILGetBool

```
int PILGetBool(char *name, int *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type boolean. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int *result [Out]`
  pointer to integer variable which will store result. Boolean value of FALSE is returned as a 0. Any other value means TRUE.

6.3.10 PILGetInt

```
int PILGetInt(char *name, int *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type integer. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int *result [Out]`
  pointer to integer variable which will store result.

6.3.11 PILGetReal

```
int PILGetReal(char *name, double *result);
```

Description

This function reads the value of specified parameter. The parameter has to be of type real. If this is not the case error code is returned.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `double *result [Out]`
  pointer to double variable which will store result.

6.3.12 PILGetReal4

```
int PILGetReal4(char *name, float *result);
```

Description

This function reads the value of specified parameter. The parameter has to be of type real. If this is not the case error code is returned.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
- name of the parameter `float *result [Out]`

pointer to float variable which will store result.

Notes

PIL library internally performs all computations using double data type. This function merely calls PILGetReal() function then converts double to float.

6.3.13 PILGetString

```
int PILGetString(char *name, char *result);
```

Description

This function reads the value of specified parameter. The parameter has to be of type string. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `char *result [Out]`
  pointer to character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

Notes

When enum list is specified for given parameter and is in effect (see section 2), then PILGetString converts value entered to uppercase before returning. Also when comparing default/entered value PILGetString ignores case. This (i.e. conversion to uppercase and case-insensitive comparison) applies _ONLY_ to string parameters and _ONLY_ to those for which pipe (vertical bar) separated enum list is specified (in `min` *field).*

6.3.14 PILGetFname

```
int PILGetFname(char *name, char *result);
```

Description

This function reads the value of specified parameter. The parameter has to be of type filename. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `char *result [Out]`
  pointer to character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

Notes
leading and trailing spaces are trimmed. If the type of the parameter specifies it, access mode checks are done on file. So if access mode specifies write mode, and the file is read-only then PILGetFname will not accept that filename and will prompt user to enter name of the file which is writable. Before applying any checks the value of the parameter (which may be URL, for instance http://, file://etc/passwd) is converted to the filename (if it is possible). Details are given in paragraph 6.3.30.

6.3.15 PILGetDOL

```
int PILGetDOL(char *name, char *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets the value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `char *result [Out]`
  pointer to character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

Notes
leading and trailing spaces are trimmed.

6.3.16 PILGetIntVector

```
int PILGetIntVector(char *name, int nelem, int *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM integer numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int nelem [In]`
  number of integers to return
- `int *result [Out]`
  pointer to vector of integers to store result

6.3.17 PILGetRealVector

```
int PILGetRealVector(char *name, int nelem, double *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM real numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int nelem [In]`
  number of doubles to return
- `double *result [Out]`
  pointer to vector of doubles to store result

6.3.18 PILGetReal4Vector

```
int PILGetReal4Vector(char *name, int nelem, float *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM real numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int nelem [In]`
  number of floats to return
- `float *result [Out]`
  pointer to vector of floats to store result

6.3.19 PILGetIntVarVector

```
int PILGetIntVarVector(char *name, int *nelem, int *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly *nelem integer numbers separated with spaces. The actual number of items found in parameter is returned in *nelem.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int *nelem [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output)
- `int *result [Out]`
  pointer to vector of integers to store result

6.3.20 PILGetRealVarVector

```
int PILGetRealVarVector(char *name, int *nelem, double *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly *nelem real numbers separated with spaces.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation. The actual number of items found in parameter is returned in *nelem.

Parameters

- `char *name [In]`
  name of the parameter
- `int *nelem [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output)
- `double *result [Out]`
  pointer to vector of doubles to store result

6.3.21 PILGetReal4VarVector

```
int PILGetReal4VarVector(char *name, int *nelem, float *result);
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have at most *nelem real numbers separated with spaces. The actual number of items found in parameter is returned in *nelem.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int *nelem [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output)
- `float *result [Out]`
  pointer to vector of floats to store result

6.3.22 PILPutBool

```
int PILPutBool(char *name, int b);
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type boolean. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int b [In]`
  new value for argument.

6.3.23 PILPutInt

```
int PILPutInt(char *name, int i);
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type integer. If this is not the case error code is returned. The same happens when value passed is out of range.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `int i [In]`
  new value for argument.

6.3.24 PILPutReal

```
int PILPutReal(char *name, double d);
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type boolean. If this is not the case error code is returned. The same happens when value passed is out of range.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `double d [In]`
  new value for argument.

6.3.25 PILPutString

```
int PILPutString(char *name, char *s);
```

Description

This function sets the value of specified parameter. without any prompts. The parameter has to be of type boolean. If this is not the case error code is returned.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `char *s [In]`
  new value for argument. Before assignment value is truncated PIL_LINESIZE characters.

6.3.26 PILPutFname

```
int PILPutFname(char *name, char *s);
```

Description

This function sets the value of specified parameter. without any prompts. The parameter has to be of type boolean. If this is not the case error code is returned. The same happens when value passed is out of range.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `char *name [In]`
  name of the parameter
- `char *s [In]`
  new value for argument. Before assignment value is truncated PIL_LINESIZE characters.

6.3.27 PILGetNumParameters

```
int PILGetNumParameters(int *parnum);
```

Description
This function returns number of parameters in parameter file. The number returned includes entries for all lines in parameter file, including those in wrong/invalid format. Actual number of valid parameters can be found by iteratively calling PILGetParameter and checking for correct format flag.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `int *parnum [Out]`
  number of parameters found

Notes
This function does not have its F90 version.

6.3.28 PILGetParameter

```
int PILGetParameter(int *parnum);
```

Description
This function returns full record about n-th parameter in parameter file. This record, stored in memory, is maintained internally by PIL. Any changes to a given parameter, are first reflected in this record and disk update is done only by PILClose or PILFlushParameters.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `idx [In]`
  index of parameter to return
- `pp [Out]`
  returned parameter's data
- `minmaxok [Out]`
  flag whether min/max(1) or enum(2) values are defined (and returned) for that parameter
- `vmin [Out]`
  min value for returned parameter (converted to proper type), only when minmaxok==1
- `vmax [Out]`
  max value for returned parameter (converted to proper type), only when minmaxok==1

Notes

One can use the following program to list parameters :

```
PILGetNumParameters(&parcnt);
for (i=0; i<parcnt; i++)
 { if (PIL_OK != PILGetParameter(i, &pardata, &minmaxflag, &minval,
                                  &maxval))
     break;
   if (PIL_FORMAT_OK != pp->format) continue;

   minmaxstr[0] = 0;
   if (1 == minmaxflag) sprintf(minmaxstr, "min=%s, max=%s ",
                                 pp->strmin, pp->strmax);
   if (2 == minmaxflag) sprintf(minmaxstr, "enum=%s ", pp->strmin);

   printf("%-16.16s 0x%02x 0x%02x %20s %s// %s\n", pp->strname,
          pp->type, pp->mode, pp->strvalue, minmaxstr, pp->strprompt);
 }
```

Symbolic values are defined in pil.h

This function does not have its F90 version.

6.3.29 PILVerifyCmdLine

```
int PILVerifyCmdLine(void);
```

Description

PILVerifyCmdLine scans argument list (given by argc and argv parameters) and for all parameters in format Name=Value checks whether there is parameter with such name in parameter table (in memory). If this is not the case it returns with an error (meaning: bogus parameters specified in command line).

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes

This function does not have its F90 version.

6.3.30 PILSetRootNameFunction

```
int PILSetRootNameFunction(int (*func)(char *s));
```

Description

This function instruct PIL to use function 'func' as a new URL to filename conversion routine. This function will be called during validation of any parameter of type file with access checking on ('fr', 'fw', 'fn' or 'fe').

PIL's default function is the one which speaks CFITSIO's language.

It is valid to call this function with func set to NULL. In this case any URL will be treated verbatim as a filename during filename validation phase.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `func [In]`
  pointer to the new URL to filename conversion function. This function should read input URL (passed in s), then it should check if it evaluates to filename. If it does it should extract that filename and put it back into s. New function should return one of the following :
  - `PIL_ROOTNAME_FILE`
    if URL evaluates to filename. For instance file://some/file.fits[1] or simply /some/file.fits. In both cases the filename stored in s will be /some/file.fits (assuming default PIL's conversion function).
  - `PIL_ROOTNAME_NOTFILE`
    if URL does not evaluate to filename
  - `PIL_ROOTNAME_STDIN`
    if URL specifies standard input stream (stdin/STDIN for instance)
  - `PIL_ROOTNAME_STDOUT`
    if URL specifies standard input stream (stdout/STDOUT for instance)
  - `PIL_ROOTNAME_STDINOUT`
    if URL specifies standard input stream ('-' for instance)
  - `other value`
    generic error code, like NULL pointers, PIL not initialized, etc...

Notes
This function does not have its F90 version.

6.3.31 PILSetReadlinePromptMode

```
int PILSetReadlinePromptMode(int mode);
```

Description
This function changes PIL's promping mode when compiled with READLINE support.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `mode [In]`
  The new PIL prompting mode. Allowable values are:

- PIL_RL_PROMPT_PIL - standard prompting mode (compatible with previous PIL versions). The prompt format is :

  ```
  some_text  [ default_value ] : X
  ```

  : (which is printed) denotes beginning of the edit buffer (which in this mode is always initially empty). X denotes cursor position. To enter empty using this mode, one has to enter "" (unless redefined by PIL_EMPTY_STRING environment variable).
- PIL_RL_PROMPT_IEB - alternate prompting mode. The prompt format is :

  ```
  some_text : default_value X
  ```

  : (which is printed) denotes beginning of the edit buffer which in this mode is initially set to the default value. X denotes cursor position. To enter empty in this mode one simply has to empty edit buffer using BACKSPACE/DEL keys then press RETURN key.

6.3.32 PILSetLoggerFunction

```
int PILSetLoggerFunction(int (*func)(char *s));
```

Description

This function instructs PIL to use function 'func' as a new logger routine. New function will be called whenever PILGetXXX or PILPutXXX routine fails. The string passed to the new routine (and generated internally by PIL) usually contains the name of the parameter for which the PIL routine failed.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `func [In]`
  pointer to the new logger function. The 'func' function should read input text (passed in s), then it should log the text somewhere using its own logging mechanism. Calling PILSetLoggerFunction(NULL) instructs PIL not to log any messages (this is also done by PILInit).

Notes

The PILSetLoggerFunction function does not have its F90 version.

7 Calling PIL library functions from C/C++

Applications can use PIL services in one of 2 different modes. The first one ISDC_SINGLE_MODE is the simplest one. In this mode application simply includes `pil.h` header file, calls PILInit, plays with parameters by calling PILGetXxx/PILPutXxx functions and finally shuts down PIL library by calling PILClose. The skeleton code is given below.

```
/* this is skeleton code for simple PIL aware applications, this
   is not a working code.
*/

#include <stdio.h>
#include <pil.h>

int main(int argc, char **argv)
{ int r;
  float fv[5];

r = PILInit(argc, argv);

if (r < 0)
  {
    printf("PILInit failed : %s\n", PIL_err_handler(r));
    return(10);
  }

r = PILGetBool("boolParName1", &intptr);
r = PILGetReal("RealParName34", &doubleptr);
r = PILGetReal4Vector("real4vecname", 5, &(fv[0]));

    /* .... application code follows .... */

PILClose(PIL_OK);
exit(0);
}
```

The second mode, called ISDC_SERVER_MODE allows for multiple rereads of parameter file. Using this method application can exchange data with other processes via parameter file (provided other processes use locks to assure exclusive access during read/write operation). One example of code is as follows :

```
/* this is skeleton code for PIL aware applications running in server mode,
   this is not a working code.
*/

#include <stdio.h>
#include <pil.h>

int main(int argc, char **argv)
{

PILInit(argc, argv);

if (ISDC_SERVER_MODE != PILRunMode)
  exit(-1);  /* error - not in server mode - check parameter file */

for (;;)
 {
   PILReloadParameters();
   PILGetInt("IntParName", &intvar);
```

```
   /* place for loop code here
        ...
   */

   PILPutReal("RealParName", 4.567);
   PILFlushParameters();
   if (exit_condition) break;
 }

PILClose(status);
return(PIL_OK);
}
```

After initial call to PILInit application jumps into main loop. In each iteration it rereads parameters from file (there is no need to call PILReloadParameters during first iteration), Based on new values of just read-in parameters (which might be modified by another process) application may decide to exit from loop or continue. If it decides to continue then after executing application specific loop code it calls PILFlushParameters to signal other process that it is done with current iteration. Algorithm described above is very simple, and the real applications are usually more complicated.

As mentioned earlier, applications written for ISDC should not use PILInit/PILClose directly. Instead they should use CommonInit/CommonExit functions from ISDC's Common Library.

Notes

most applications will not support ISDC_SERVER_MODE so one can delete those fragments of skeleton code which deal with this mode.

Part III

PIL Fortran 90 Language Interface

8 PIL F90 Language API

8.1 Introduction

This section describes the Fortran 90 language implementation of the Parameter Interface Library Application Programming Interfaces (PIL APIs) It also gives Fortran 90 languages specific information necessary to use those APIs. Users interested only in using PIL from C/C++ applications should see section 6.

PIL library is standalone and can be compiled independently of other ISDC libraries.

Unless stated otherwise all PIL API functions return status code of type int. This is either ISDC_OK which equals PIL_OK which equals 0, which means everything went perfectly or negative value (error code) meaning some error occured. List of error codes can be found in pil_f90_api.f90 file. Functions given below are the "official" ones. Internally PIL library calls many more functions.

8.2 PIL Fortran 90 module files

Applications calling PIL services from Fortran 90 source code should utilize

```
USE PIL_F90_API
```

statement to include all PIL definitions. Alternatively they can

```
USE ISDC_F90_API
```

which internally includes PIL_F90_API module.

PIL_F90_API compiled module contains prototypes of all Fortran 90 API functions, definitions of constants, declarations of global variables and definitions of data structures.

8.3 Fortran 90 API functions

8.3.1 PILINIT

```
FUNCTION PILINIT()
INTEGER :: PILINIT
```

Description

This function initializes PIL library. This function has to be called before any other PIL function (there are some exceptions to this rule). It does the following :
First it calculates name of parameter file. Usually name of parameter file equals GETARG(0)+ ".par" suffix but this can be overriden by calling PILSETMODULENAME and/or PILSETMODULEVERSION before calling PILINIT. After successful termination parameter file is opened and read in, and global variable PILRunMode is set to one of the following values :

```
ISDC_SINGLE_MODE
ISDC_SERVER_MODE
```

ISDC_SERVER_MODE is set whenever there is parameter "ServerMode" and its value is "yes" or "y". In any other case PILRunMode is set to ISDC_SINGLE_MODE.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes

Only one parameter file can be open at a time. Parameter file remains open until PILCLOSE is called. When writing applications for ISDC one should not use PILINIT directly. Instead, one should call COMMONINIT function (from Common library) which calls PILINIT.

8.3.2 PILCLOSE

```
FUNCTION PILCLOSE(STATUS)
INTEGER*4 :: STATUS
INTEGER :: PILCLOSE
```

Description

This function has to be called before application terminates. It closes open files, and writes all learned parameters to the disk files (only when status == ISDC_OK). Once this function is called one cannot call any other PIL functions. One can however call PILINIT to reinitialize PIL library. Function also clears PILRunMode, PILModuleName and PILModuleVersion global variables. Those global variables are directly accessible from F90 code.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `INTEGER*4 :: STATUS [In]`
  PIL_OK/ISDC_OK/0 means normal return, any other value means abnormal termination. In this case changes to parameters made during runtime are NOT written to parameter files.

Notes

this function does not terminate process. It simply shuts down PIL library. When writing applications for ISDC one should not use PILCLOSE directly. Instead, one should call COMMONEXIT function (from Common library) which calls PILCLOSE.

8.3.3 PILRELOADPARAMETERS

```
FUNCTION PILRELOADPARAMETERS()
INTEGER :: PILRELOADPARAMETERS
```

Description

This function reloads parameters from parameter file. It is called internally by PILINIT. It should be called explicitly by applications running in ISDC_SERVER_MODE to rescan parameter file and reload parameters from it. Current parameter list in memory (including any modifications) is deleted. PIL library locks whole file for exclusive access when reading from parameter file.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes
parameter file remains open until PILCLOSE is called. Function internally DOES NOT close/reopen parameter file.

8.3.4 PILFLUSHPARAMETERS

```
FUNCTION PILFLUSHPARAMETERS()
INTEGER :: PILFLUSHPARAMETERS
```

Description
This function flushes changes made to parameter list (in memory) to disk. Current contents of parameter file is overwritten. PIL library locks whole file for exclusive access when writing to parameter file.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Notes
parameter file remains open until PILCLOSE is called.

8.3.5 PILSETMODULENAME

```
FUNCTION PILSETMODULENAME(NAME)
CHARACTER*(*) :: NAME
INTEGER :: PILSETMODULENAME
```

Description
Sets name of the module which uses PIL services. Result is stored in global variable PILModuleName. Usually name of parameter file equals argv[0] + ".par" suffix but this can be overriden by calling PILSETMODULENAME and/or PILSETMODULEVERSION before calling PILINIT.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  new module name

8.3.6 PILSETMODULEVERSION

```
FUNCTION PILSETMODULEVERSION(VERSION)
CHARACTER*(*) :: VERSION
INTEGER :: PILSETMODULEVERSION
```

Description
Sets version of the module which uses PIL services. Result is stored in global variable PILModuleVersion. If NULL pointer is passed version is set to "version unspecified"

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: VERSION [In]`
  new module version (string)

8.3.7 PILGETPARFILENAME

```
FUNCTION PILGETPARFILENAME(FNAME)
CHARACTER*(*) :: FNAME
INTEGER :: PILGETPARFILENAME
```

Description
This function retrieves full path of used parameter file. Absolute path is returned only when PFILES environment variable contains absolute paths. If parameter file is taken from current dir then only filename is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: FNAME [Out]`
  name/path of the parameter file in use

Notes
FNAME buffer should be at least PIL_LINESIZE characters long.

8.3.8 PILOVERRIDEQUERYMODE

```
FUNCTION PILOVERRIDEQUERYMODE(NEWMODE)
INTEGER*4 :: NEWMODE
INTEGER :: PILOVERRIDEQUERYMODE
```

Description
This functions globally overrides query mode. When newmode passed is PIL_QUERY_OVERRIDE, prompting for new values of parameters is completely disabled. If value is bad or out of range PILGETXXX return immediately with error without asking user. No i/o in stdin/stdout is done by PIL in this mode. When newmode is PIL_QUERY_DEFAULT PIL reverts to default query mode.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `INTEGER*4 :: NEWMODE [In]`
  new value for query override mode

8.3.9 PILGETBOOL

```
FUNCTION PILGETBOOL(NAME, RESULT)
CHARACTER*(*) :: NAME
INTEGER*4 :: RESULT
INTEGER :: PILGETBOOL
```

Description
This function reads the value of specified parameter. The parameter has to be of type boolean. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER*4 :: RESULT [Out]`
  integer variable which will store result. Boolean value of FALSE is returned as a 0. Any other value means TRUE.

8.3.10 PILGETINT

```
FUNCTION PILGETINT(NAME, RESULT)
CHARACTER*(*) :: NAME
INTEGER*4 :: RESULT
INTEGER :: PILGETINT
```

Description
This function reads the value of specified parameter. The parameter has to be of type integer. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER*4 :: RESULT [Out]`
  integer variable which will store result.

8.3.11 PILGETREAL

```
FUNCTION PILGETREAL(NAME, RESULT)
CHARACTER*(*) :: NAME
REAL*8 :: RESULT
INTEGER :: PILGETREAL
```

Description
This function reads the value of specified parameter. The parameter has to be of type real. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `REAL*8 :: RESULT [Out]`
  REAL*8 variable which will store result.

8.3.12 PILGETREAL4

```
FUNCTION PILGETREAL4(NAME, RESULT)
CHARACTER*(*) :: NAME
REAL*4 :: RESULT
INTEGER :: PILGETREAL4
```

Description
This function reads the value of specified parameter. The parameter has to be of type real. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `REAL*4 :: RESULT [Out]`
  REAL*4 variable which will store result.

Notes
*Function merely calls PILGETREAL, then converts REAL*8 to REAL*4*

8.3.13 PILGETSTRING

```
FUNCTION PILGETSTRING(NAME, RESULT)
CHARACTER*(*) :: NAME
CHARACTER*(*) :: RESULT
INTEGER :: PILGETSTRING
```

Description
This function reads the value of specified parameter. The parameter has to be of type string. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `CHARACTER*(*) :: RESULT [Out]`
  character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

8.3.14 PILGETFNAME

```
FUNCTION PILGETFNAME(NAME, RESULT)
CHARACTER*(*) :: NAME
CHARACTER*(*) :: RESULT
INTEGER :: PILGETFNAME
```

Description
This function reads the value of specified parameter. The parameter has to be of type filename. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `CHARACTER*(*) :: RESULT [Out]`
  character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

Notes
leading and trailing spaces are trimmed. If type of parameter specifies it, access mode checks are done on file. So if access mode specifies write mode, and file is read-only then PILGETFNAME will not accept this filename and will prompt user to enter name of writable file. Before applying any checks the value of the parameter (which may be URL, for instance http://, file://etc/passwd) is converted to the filename (if it is possible). Details are given in paragraph 6.3.30.

8.3.15 PILGETDOL

```
FUNCTION PILGETDOL(NAME, RESULT)
CHARACTER*(*) :: NAME
CHARACTER*(*) :: RESULT
INTEGER :: PILGETDOL
```

Description
This function reads the value of specified parameter. The parameter has to be of type string. If this is not the case error code is returned. It is possible to enter empty string (without accepting default value). By default entering string "" (two doublequotes) sets value of given string parameter to an empty string. If PIL_EMPTY_STRING environment variable is defined then its value is taken as an empty string equivalent.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `CHARACTER*(*) :: RESULT [Out]`
  character array which will store result. The character array should have at least PIL_LINESIZE characters to assure enough storage for the longest possible string.

Notes
leading and trailing spaces are trimmed

8.3.16 PILGETINTVECTOR

```
FUNCTION PILGETINTVECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
INTEGER*4 :: RESULT
INTEGER   :: PILGETINTVECTOR
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM integer numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER :: NELEM [In]`
  number of integers to return
- `INTEGER*4 :: RESULT [Out]`
  pointer to vector of integers to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

8.3.17 PILGETREALVECTOR

```
FUNCTION PILGETREALVECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
REAL*8    :: RESULT
INTEGER   :: PILGETREALVECTOR
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM REAL*8 numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER :: NELEM [In]`
  number of integers to return
- `REAL*8 :: RESULT [Out]`
  pointer to vector of REAL*8's to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

8.3.18 PILGETREAL4VECTOR

```
FUNCTION PILGETREAL4VECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
REAL*4    :: RESULT
INTEGER   :: PILGETREAL4VECTOR
```

Description

This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly NELEM REAL*4 numbers separated with spaces. If there are more or less then NELEM then error code is returned.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER :: NELEM [In]`
  number of integers to return
- `REAL*4 :: RESULT [Out]`
  pointer to vector of REAL*4's to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

Notes

*Function merely calls PILGETREAL, then converts REAL*8 to REAL*4*

8.3.19 PILGETINTVARVECTOR

```
FUNCTION PILGETINTVARVECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
INTEGER*4 :: RESULT
INTEGER   :: PILGETINTVARVECTOR
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly *nelem integer numbers separated with spaces. The actual number of items found in parameter is returned in *nelem.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER :: NELEM [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output) `INTEGER*4 :: RESULT [Out]`
  pointer to vector of integers to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

8.3.20 PILGETREALVARVECTOR

```
FUNCTION PILGETREALVARVECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
REAL*8    :: RESULT
INTEGER   :: PILGETREALVARVECTOR
```

Description
This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly *nelem integer numbers separated with spaces. The actual number of items found in parameter is returned in *nelem.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER :: NELEM [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output) `REAL*8 :: RESULT [Out]`
  pointer to vector of REAL*8's to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

8.3.21 PILGETREAL4VARVECTOR

```
FUNCTION PILGETREAL4VARVECTOR(NAME, NELEM, RESULT)
CHARACTER*(*) :: NAME
INTEGER   :: NELEM
REAL*4    :: RESULT
INTEGER   :: PILGETREAL4VARVECTOR
```

Description

This function reads the value of specified parameter. The parameter has to be of type string (type stored in parameter file). If this is not the case error code is returned. Ascii string (value of parameter) has to have exactly *nelem integer numbers separated with spaces. The actual number of items found in parameter is returned in *nelem.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) ::  NAME [In]`
  name of the parameter
- `INTEGER ::  NELEM [In/Out]`
  Maximum expected number of item (on input). Actual number of floats found in parameter (on output) `REAL*4 ::  RESULT [Out]`
  pointer to vector of REAL*4's to store result. Actually, one should pass 1st element of vector say VECNAME(1), and not VECNAME

Notes

*Function merely calls PILGETREAL, then converts REAL*8 to REAL*4*

8.3.22 PILPUTBOOL

```
FUNCTION PILPUTBOOL(NAME, VALUE)
CHARACTER*(*) :: NAME
INTEGER*4 :: VALUE
INTEGER :: PILPUTBOOL
```

Description

This function sets the value of specified parameter. without any prompts. The parameter has to be of type boolean. If this is not the case error code is returned.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER*4 :: VALUE [In]`
  new value for argument. 0 means FALSE, any other value means TRUE

8.3.23 PILPUTINT

```
FUNCTION PILPUTINT(NAME, VALUE)
CHARACTER*(*) :: NAME
INTEGER*4 :: VALUE
INTEGER :: PILPUTINT
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type integer. If this is not the case error code is returned. The same happens when value passed is out of range.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `INTEGER*4 :: VALUE [In]`
  new value for argument.

8.3.24 PILPUTREAL

```
FUNCTION PILPUTREAL(NAME, VALUE)
CHARACTER*(*) :: NAME
REAL*8 :: VALUE
INTEGER :: PILPUTREAL
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type real. If this is not the case error code is returned. The same happens when value passed is out of range

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter

- `REAL*8 :: VALUE [In]`
  new value for argument.

8.3.25 PILPUTSTRING

```
FUNCTION PILPUTSTRING(NAME, VALUE)
CHARACTER*(*) :: NAME
CHARACTER*(*) :: VALUE
INTEGER :: PILPUTSTRING
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type string. If this is not the case error code is returned.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `CHARACTER*(*) :: VALUE [In]`
  new value for argument. Before assignment value is truncated PIL_LINESIZE characters.

8.3.26 PILPUTFNAME

```
FUNCTION PILPUTFNAME(NAME, VALUE)
CHARACTER*(*) :: NAME
CHARACTER*(*) :: VALUE
INTEGER :: PILPUTFNAME
```

Description
This function sets the value of specified parameter. without any prompts. The parameter has to be of type filename. If this is not the case error code is returned. The same happens when value passed is out of range.

Return Value
If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `CHARACTER*(*) :: NAME [In]`
  name of the parameter
- `CHARACTER*(*) :: VALUE [In]`
  new value for argument. Before assignment value is truncated PIL_LINESIZE characters.

8.3.27 PILSETREADLINEPROMPTMODE

```
FUNCTION PILSETREADLINEPROMPTMODE(MODE)
INTEGER :: MODE
```

Description

This function changes PIL's promping mode when compiled with READLINE support.

Return Value

If success returns ISDC_OK. Error code otherwise. See appendix A for a list of error codes and their explanation.

Parameters

- `mode [In]`
  The new PIL prompting mode. Allowable values are:
  - PIL_RL_PROMPT_PIL - standard prompting mode (compatible with previous PIL versions). The prompt format is :

    ```
    some_text  [ default_value ] : X
    ```

    : (which is printed) denotes beginning of the edit buffer (which in this mode is always initially empty). X denotes cursor position. To enter empty using this mode, one has to enter "" (unless redefined by PIL_EMPTY_STRING environment variable).
  - PIL_RL_PROMPT_IEB - alternate prompting mode. The prompt format is :

    ```
    some_text : default_value X
    ```

    : (which is printed) denotes beginning of the edit buffer which in this mode is initially set to the default value. X denotes cursor position. To enter empty in this mode one simply has to empty edit buffer using BACKSPACE/DEL keys then press RETURN key.

9 Calling PIL library functions from Fortran 90

Applications can use PIL services in one of 2 different modes. The first one ISDC_SINGLE_MODE is the simplest one. In this mode application simply includes PIL definitions from compiled module file (`USE PIL_F90_API` statement), calls PILINIT, plays with parameters by calling PILGETXXX/PILPUTXXX functions and finally shuts down PIL library by calling PILCLOSE. The skeleton code is given below.

```
PROGRAM SKELETON

USE PIL_F90_API

integer :: r
REAL*8  :: real8vec(5)

r = PILINIT()
if (ISDC_OK /= r) STOP
```

```
r = PILGETBOOL("boolname1", INTVAR)
r = PILGETREAL("realname3", REALVAR)
r = PILGETREALVECTOR("real8vecname", 5, real8vec(1))

! now execute application code ...
r = PILEXIT(ISDC_OK)

END
```

The second mode, called ISDC_SERVER_MODE allows for multiple rereads of parameter file. Using this method application can exchange data with other processes via parameter file (provided other processes use locks to assure exclusive access during read/write operation). One example of code is as follows :

```
PROGRAM SKELETON2

USE PIL_F90_API

integer :: r

r = PILINIT()
if (ISDC_OK /= r) STOP

DO
  r = PILRELOADPARAMETERS()
  r = PILGETBOOL("boolname1", INTVAR)
  r = PILGETREAL("realname3", REALVAR)
  IF (BREAKCONDITION) BREAK

! now execute loop code ...

  r = PILPUTINT("intname45", INTVAR)
  r = PILFLUSHPARAMETERS()

  IF  (BREAKCONDITION) BREAK

ENDDO

r = PILEXIT(ISDC_OK)
END
```

After initial call to PILINIT application jumps into main loop. In each iteration it rereads parameters from file (there is no need to call PILRELOADPARAMETERS during first iteration), Based on new values of just read-in parameters (which might be modified by another process) application may decide to exit from loop or continue. If it decides to continue then after executing application specific loop code it calls PILFLUSHPARAMETERS to signal other process that it is done with current iteration. Algorithm described above is very simple, and it real applications can be much more complicated.

As mentioned earlier, applications written for ISDC should not use PILINIT/PILCLOSE directly. Instead they should use COMMONINIT/COMMONEXIT functions from ISDC's Common Library.

Notes

most applications will not support ISDC_SERVER_MODE so one can delete those fragments of skeleton code which deal with this mode.

Part IV

Appendices

A PIL Error Codes

Error codes (symbolic names and values) are common to both C/C++ and Fortran 90 API. Whenever text in the following table mentions C/C++ function, the same holds for its Fortran 90 equivalent.

Error Symbol	**Code**	**Error Description**
ISDC_OK	0	No error
PIL_OK	0	No error (alias of ISDC_OK)
PIL_NUL_PTR	-3000	Null pointer passed as an argument, and function does not allow it. Can appear in many situations. For example passing NULL as a name of argument or passing NULL as a pointer to result variable causes PILGetXXX function to return this error code.
PIL_BAD_ARG	-3001	Bad argument passed. This error may be returned in many situations. Usually means that data type is incorrect, i.e. calling PILGetInt("a",..) when parameter "a" is of type string. Also when check for file access mode fails (PILGetFname) this error is returned. When PIL_QUERY_OVERRIDE mode is in effect and default value of parameter is out of range or invalid this error is returned (PILGetXXX) This error may also mean internal PIL error.
PIL_NO_MEM	-3002	Not enough memory. Means that PIL library was not able to allocate sufficient memory to handle request. This kind of error is very unlikely to happen, since PIL allocates very small amounts of memory. If it happens it probably means that memory is overwritten by process or machine (operating system) is not in good shape.
PIL_NO_FILE	-3003	Cannot open/create file. This error may be returned by PILInit when parameter file cannot be found/created/copied/updated and by PILgetFname when testing for files existence.
PIL_ERR_FREAD	-3004	Read from file failed. Means physical disk i/o error, lock problem, or network problem when parameter file resides on NFS mounted partition. This error may be returned by PILInit, PILFlushParameters, PILReloadParameters, PILClose
PIL_ERR_FWRITE	-3005	Write to file failed. Means physical disk i/o error, disk full, lock problem, or network problem when parameter file resides on NFS mounted partition. This error may be returned by PILInit, PILFlushParameters, PILReloadParameters, PILClose

Error Symbol	**Code**	**Error Description**
PIL_EOS	-3006	Unexpected end of string. This error means that given line in parameter file is badly formatted and does not contain 7 fields separated by ','. This error is handled internally by PIL library and should never be returned to user, since PIL library ignores all badly formatted lines in parameter files.
PIL_BAD_NAME	-3007	Invalid name of parameter. This error means that name of parameter found in parameter file contains invalid characters (ASCII-7 only are allowed).
PIL_BAD_TYPE	-3008	Invalid type of parameter in parameter file. This error means that PIL was not able to decode parameter type. Probably parameter file is badly formatted.
PIL_BAD_MODE	-3009	Invalid mode of parameter in parameter file. This error means that PIL was not able to decode parameter mode. Probably parameter file is badly formatted.
PIL_BAD_LINE	-3010	Invalid line in parameter file encountered. This error means that format of line in parameter file does not even resemble correct one.
PIL_NOT_IMPLEMENTED	-3011	Feature not implemented. This error means that type/mode/indirection type is valid (according to IRAF/XPI standard) but PIL library does not implement this. Example of unimplemented feature is indirection of data from other parameter files.
PIL_FILE_NOT_EXIST	-3012	File does not exist. Included for compatibility reasons. Version 1.0 of PIL does not return this error.
PIL_FILE_EXIST	-3013	File exists. Included for compatibility reasons. Version 1.0 of PIL does not return this error.
PIL_FILE_NO_RD	-3014	File is not readable. Check file access permission/ownership and mount options.
PIL_FILE_NO_WR	-3015	File is not writable. Check file access permission/ownership and mount options. By default PIL requires READWRITE access to the parameter file.
PIL_LINE_BLANK	-3016	Blank line encountered. Handled internally by PIL. Never returned to user.
PIL_LINE_COMMENT	-3017	Comment line encountered. Handled internally by PIL. Never returned to user.
PIL_LINE_ERROR	-3018	Invalid line encountered. Handled internally by PIL. Never returned to user.

Error Symbol	**Code**	**Error Description**
PIL_NOT_FOUND	-3019	No such parameter. This error means that no parameter of given name is in parameter file.
PIL_PFILES_TOO_LONG	-3020	PFILES environment variable too long. Included for compatibility reasons. Version 1.0 of PIL does not return this error.
PIL_PFILES_FORMAT	-3021	PFILES environment variable is badly formatted. Included for compatibility reasons. Version 1.0 of PIL does not return this error.
PIL_LOCK_FAILED	-3022	Cannot (un)lock parameter file. This error means that PIL library couldn't obtain exclusive access to file (by locking it). Operation still was performed, by consistency of data od disk is not assured.
PIL_BOGUS_CMDLINE	-3023	Bogus parameters found in command line. In other words some of parameter names specified in command line do not have their counterparts in parameter file. This error may be returned by PILVerifyCmdLine().
PIL_NO_LOGGER	-3024	Error code used internally by PIL to signal that no logger function has been registered.
PIL_LINE_TOO_MANY	-3025	Format error in parameter file. Too many (¿7) comma separated items found.
PIL_LINE_TOO_FEW	-3026	Format error in parameter file. Too few (¡7) comma separated items found.
PIL_LINE_UNMATCHED_QUOTE	-3027	Format error in parameter file. Unbalanced quote or doublequote character found.
PIL_LINE_NO_LF	-3028	Format error in parameter file. No terminating LineFeed character found.
PIL_LINE_EXTRA_SPACES	-3029	Format error in parameter file. Extra spaces following closing quote or doublequote character found. found.
PIL_BAD_VAL_BOOL	-3030	Cannot convert string to boolean value. Expecting either yes or no string.
PIL_BAD_VAL_INT	-3031	Cannot convert string to integer value. Expecting [+-]dddddd format
PIL_BAD_VAL_REAL	-3032	Cannot convert string to real value. Expecting [+-]ddd[.ddd][e[+-ddd]] format
PIL_BAD_VAL_INT_VAR_VECTOR	-3033	Cannot convert string to a vector of integers. Expecting space separated list of integer numbers.

PIL_BAD_VAL_INT_VECTOR	-3034	Cannot convert string to a vector of integers. Expecting space separated list of integer numbers. This error is also returned when number of integers found in the string does not correspond to the number given in a call to PILGetIntVector();
PIL_BAD_VAL_REAL_VAR_VECTOR	-3035	Cannot convert string to a vector of reals. Expecting space separated list of real numbers.
PIL_BAD_VAL_REAL_VECTOR	-3036	Cannot convert string to a vector of reals. Expecting space separated list of real numbers. This error is also returned when number of reals found in the string does not correspond to the number given in a call to PILGetRealVector() or PILGetReal4Vector().
PIL_OFF_RANGE	-3037	Value is off limits set by min and max fields. This error may be returned when PIL_QUERY_OVERRIDE mode is in effect.
PIL_BAD_ENUM_VALUE	-3038	Value does not match any value from enumerated list (min field). This error may be returned when PIL_QUERY_OVERRIDE mode is in effect.
PIL_BAD_FILE_ACCESS	-3039	Effective file access mode does not match the one specified in parameter type (i.e. file is not readable when type is "fr"). This error may be returned when PIL_QUERY_OVERRIDE mode is in effect.