SDP Toolkit Primer for the ECS Project

1.Introduction

1.1 Purpose of Toolkit

The purpose of the SDP Toolkit is primarily (1) to provide an interface to the ECS system, including Planning and Data Production System (PDPS), Communications System Management (CSMS) and Information Management, (2) to allow science software to be portable to different platforms at the DAAC, (3) to reduce redundant coding at the SCF, and (4) to provide value added functionality for science software development. The SDP Toolkit consists of a set of fully tested, fast, efficient and reliable C and FORTRAN language functions, customized for application to ECS.

A brief overview of the operations concept of the Toolkit follows. The Toolkit divides into two groups: Mandatory tools, which the system requires in science software, with checking to occur at DAAC Integration & Test time; and Optional tools, whose primary intention is to save SCF development effort by reducing redundancy.

1.2. Mandatory Tools

The following tools are Mandatory:

At the lowest level are the Error and Status Message (**SMF**, for Status Message Facility) tools, which provide general error handling, status log messaging, and interface to CSMS services (which are implemented as email and ftp services at the SCF). Essentially all Toolkit functions call the SMF tools for error handling; science software may also use most of the SMF functions. (The Toolkit takes no action itself regarding errors itself; this is left to the science software.)

At the next level are the Process Control (**PC**) tools, which provide the primary interface to the Planning and Data Production System (PDPS). A major use of these tools is to access physical filenames and file attributes; in addition, they retrieve user-defined parameters. Several Toolkit functions call PC tools.

Generic Input/Output (**IO_Gen**) tools are at the next level; these tools provide the means to open and close support, temporary and intermediate duration files. Native C and FORTRAN functions perform the actual reads and writes.

Memory allocation (**MEM**) tools consist of two groups: the first consists of simple wrappers on native C functions, the purpose being to track memory usage in the SDPS; the second consists of "shared memory" tools, which enable the sharing of memory among executables within a PGE.

The rest of the Mandatory tools are higher level, in that they depend on at least some of the lower level tools:

Level 0 access (**IO_L0**) tools access Level 0 data.

Metadata (**MET**) access tools allow science software to access, alter, write and append metadata.

Spacecraft ephemeris and attitude access (**EPH**) tools read ephemeris and attitude data.

Time and Date (**TD**) tools perform time and date conversions between selected time systems.

1.3 Optional Tools

The remaining tools are Optional:

Ancillary data Access (**AA**) functions access such data as NMC data and Digital Elevation (DEM) data.

Celestial Body Position (**CBP**) tools locate the Sun, the Moon and the planets.

Coordinate System Conversion (**CSC**) tools allow coordinate conversions between celestial reference, spacecraft body referenced, spacecraft orbital referenced, and Earth frames. They also perform related tasks such as locating the sub-satellite point (ground track) and finding the zenith and azimuth of vectors at Earth surface.

Constant and Unit Conversion (**CUC**) tools allow access to physical constants and unit conversions.

Digital Elevation Model (**DEM**) tools provide access to HDF-EOS DEM datasets. This will be the primary production DEM data.

The IMSL package provides mathematical and statistical support.

Graphics Support (if any, in the production environment) is TBD.

There are also some Test Tools, which are for use during development at the SCF only. These include an ephemeris and attitude simulator and a Level 0 file simulator.

For the most part the Optional tools are independent of each other, though all depend on the lower level tools, including SMF (all tools), PC, IO_Gen, and TD.

1.4 Toolkit Languages

The Toolkit is written in the C language. A macro package provides bindings to the C code from FORTRAN 77 (with a few exceptions coded directly in F77). These bindings appear to have no effect on processing speed. Where possible, the same Application Program Interface (API), i.e., calling sequence, has been used for both C and FORTRAN. Support of FORTRAN 90 requires no special bindings, since FORTRAN 77 is a subset of FORTRAN 90; testing the Toolkit with an F90 compiler confirms this.

Special note regarding FORTRAN: Programmers are strongly urged to include the IMPLICIT NONE statement at the beginning of every FORTRAN module. This prevents many types of error; in particular, there is less chance you could omit an include file needed for a Toolkit function.

1.5 Purpose of This Document

This document refers to those functions delivered as of the Release B.0 SCF Toolkit (April 1997). Each successive delivery increments the previous delivery with additional functionality, while maintaining a consistent user API. The document will be updated with each successive software delivery.

A user's guide (*Release B.0 SCF Toolkit Users Guide, April1997*) accompanies the Toolkit delivery. The intent of this guide was to serve as the sole documentation for use of the Toolkit. However, after review, several instrument teams pointed out that it was not useful as a simple introduction to the Toolkit; rather, it resembled the detail and complexity of Unix "man" pages. This document intends to fill that gap.

The purpose of this document is to provide a simple, easy to use guide to Toolkit function usage, through a step-by-step format, including many examples in C and FORTRAN. The intended audience is both science software programmers and their supervisors. After reading it, the user will be able to use the Toolkit API in constructing instrument data production code or incorporating Toolkit calls into heritage code.

This document is necessarily not a comprehensive one; the *TK5.2 version of the Users Guide* is the definitive source. It contains details such as Toolkit installation instructions, requirements trace, detailed description of inputs and output data and parameters, and so on. For purposes of this document, we assume that the user has a copy of the Toolkit already installed on his/her system, including especially the setting of Toolkit environment variables.

1.6 Document Format

Each of the tool groups delivered to date is listed in its own section. An overview sub-section explains the general usage of the tool group. For each tool, we include: a short explanation of what it's for; step-by-step guide to usage by example, for C and FORTRAN; and a Notes section which includes dependencies on other Toolkit functions, files and environment variables. The examples given are for illustrative purposes only; for compilable examples, please refer to the software test drivers that are part of the Toolkit delivery package.

The Status/Message (SMF), Process Control (PC), and Ancillary Data Access (AA) tool groups are exceptions to the format, in that they need extensive explanation regarding their use as a whole; their "Overview" sections are very long.

2. Error and Status Message (SMF) Tools

2.1 Overview

2.1.1 Introduction

The Error/Status Message (a.k.a. SMF, for "Status Message Facility") Tools are the lowest level of the Toolkit, since nearly all of the other Toolkit functions call these tools. Their purpose is to provide an error and status message handling mechanism for use in science software (and in Toolkit functions), and to provide means to send log files, informational messages and output data files to DAAC personnel or to remote users.

In this overview section, we walk you through the procedure of constructing your own error/status messages step-by-step, then show their application in log files, your own code, and in the Toolkit itself.

2.1.2 Constructing Your Own Error/Status Messages

This section explains how to use the Toolkit to construct files containing error and status messages, which your code can access at runtime.

The basic process of constructing these files consists of 2 steps: constructing the status message text file with an editor, then running the smfcompile utility provided in the Toolkit, before compiling and executing your code.

2.1.2.1 The status message text file

The first step is to type in your own messages into the **status message text file** using a text editor.

You may use as many status message text files as you like, **provided you use a different seed number for each file** (see "%SEED" field below). For purposes of internal Toolkit efficiency, it is recommended that each set of error messages that correspond to a given set of modules in your code be defined in a separate file -- it is not efficient to mix them across module groupings, nor to put them all in one big file. For example, all messages pertaining to your geolocation processing might be in one file, and all related to ancillary data processing in another.

These files always have the suffix ".t". We present an example of this file, adapted from a prototype of the Toolkit that uses heritage AVHRR/Land Pathfinder code from GSFC. This file is also given in Appendix A of this document. These messages are examined further in the following sections.

```
# Status Message Text File for Toolkit AVHRR/Land Pathfinder
# prototype
#
%INSTR        = AVHRR
%LABEL        = PATHFINDER
%SEED         = 99
PATHFINDER_F_OPEN_BINARY_FILE   FATAL_ERROR...opening binary file
PATHFINDER_F_MEM_ALLOC_FAIL     FATAL ERROR...allocating memory %s
PATHFINDER_F_OPEN_ANC_FILE      FATAL_ERROR...%s
PATHFINDER_W_CLOSE_GAC_FILE     WARNING...could not close GAC file
PATHFINDER_W_OZONE_FILE_MISSING        Ozone file not found
                                      ::PATHFINDER_A_ALT_FILE_USED
PATHFINDER_A_ALT_FILE_USED      Alternate file used
PATHFINDER_W_EPH_FILE_NOT_FOUND        Ephemeris file not found
                                      ::PATHFINDER_A_ALT_FILE_USED
PATHFINDER_W_NO_LOG_FILES       WARNING: Problem sending log files
PATHFINDER_N_PROCESSING_DONE    SUCCESS: processing complete at %s
```

2.1.2.2 Constructing the status message text file header

The first 3 lines of this file are comments. The next 3 lines are required. They may appear only once per file, and must appear in this order.

```
%INSTR        = AVHRR
```

The "%INSTR" field is your **instrument name**.

```
%LABEL        = PATHFINDER
```

The **label** in the "%LABEL" field is arbitrary (see **label** below in this section) .

Both of the above fields must consist of 3 to 10 upper case letters.

```
%SEED         = 99
```

The "%SEED" field is a **seed number** assigned to you by ECS/SDPS. Most teams have been allocated 5,000 seed values in a specified range. The purpose of seed numbers is to ensure unique error messages for each instrument team or development group.

Given the example here, the name of the status message text file containing all of this information is recommended to be "AVHRR_99.t".

2.1.2.3 Constructing the status definitions: Simple message

```
PATHFINDER_F_OPEN_BINARY_FILE   FATAL_ERROR...opening binary file
```

The remainder of the file contains the definition of your error and status messages. Each consists of a single **status definition**, of which there may be up to 510 per file. (If you need more, just make another file with a new "%LABEL".) Status definitions may span several lines, as whitespace is ignored. Each status definition consists of two parts.

```
PATHFINDER_F_OPEN_BINARY_FILE FATAL_ERROR...opening binary file
```

The first part, the **mnemonic label**, is what you will pass to the error/status reporting functions in your code. It consists of 3 tokens, and may consist of **up to 30 uppercase letters and underscores**.

```
PATHFINDER_F_OPEN_BINARY_FILE FATAL_ERROR...opening binary file
```

The first token in the mnemonic label must be identical to the "%LABEL" field, i.e., the **label**. This provides the means to separate messages by functional groups in the science software -- each group would have its own status message (".t") file, with the "%LABEL" field providing the group ID.

```
PATHFINDER_F_OPEN_BINARY_FILE  FATAL_ERROR...opening binary file
```

The second token in the mnemonic label is the **status level**. The following table contains a list of the possible levels. The order listed in this table is significant.

```
Table. Error/Status Message Levels.
Level Name Description
_S_ Success Normal return value
_A_ Action For retrieving a string indicating action taken
_M_ Message Message returned by Toolkit
_U_ User information Informational message generated by user
_N_ Notice E.g., for data availability notices
_W_ Warning Possible problem in program
_E_ Error Error in program
_F_ Fatal error Fatal error in program
```

In our example the level of the message is "_F_", or fatal error. Note that the Toolkit itself takes no action based on the status level; that is the province of the science software. See the PGS_SMF_Test*Level(sec. 2.2.12) tool sub-group in the Tool Description section for an explanation of how to utilize these levels. (Note that "Action" is not a valid status level.)

```
PATHFINDER_F_OPEN_BINARY_FILE  FATAL_ERROR...opening binary file
```

The third token in the mnemonic label indicates the content of the message.

```
PATHFINDER_F_OPEN_BINARY_FILE  FATAL_ERROR...opening binary file
```

The second part of the line entry in the status message text file, the **message string**, is the actual text that gets printed. It consists of up to 240 ASCII characters. Any whitespace is reduced to a single space.

What happens to the entries in the status message text file is the subject of the next section. A few more examples of status message text file entries are in order first.

2.1.2.4 Constructing the status definitions: Message with runtime value added

```
PATHFINDER_F_MEM_ALLOC_FAIL   FATAL ERROR...allocating memory %s
```

This example shows the possibility of adding the value of a variable to a message string, through the C language format specifier %s. See PGS_SMF_SetDynamicMsg(sec. 2.2.9) in the Tool Description section for the method for doing this. (The FORTRAN 77 implementation of this is under study at this writing.)

2.1.2.5 Constructing the status definitions: Action message

```
PATHFINDER_A_ALT_FILE_USED                Alternate file used
```

This example shows how to implement action messages in the status message text file. **Action messages** are simply a convenient way to specify in the status messages the action taken in response to a condition. It is easiest to explain this by example.

```
PATHFINDER_W_OZONE_FILE_MISSING        Ozone file not found ::PATHFINDER_A_ALT_FILE_USED
```

(**Note:** All of the above must appear on a single line)

Above is an example **action definition**.

```
PATHFINDER_A_ALT_FILE_USED
```

is the **action label**, with level "_A_". If in the course of processing your program tries to open the ozone file, but does not find it, then it may set the warning message

PATHFINDER_W_OZONE_FILE_MISSING.

The Toolkit then writes the string "Ozone file not found" to the Status log file. (See section 2.1.3, "Log files", for explanation of different log files.) You might want the response you take to be written to a log file, using a pre- defined message; this can be done using the action definition. If your lower level module returned PATHFINDER_W_OZONE_FILE_MISSING, then you can call Toolkit function PGS_SMF_GetActionByCode with this mnemonic as input, and get back the string "Alternate file used". You could then use the PGS_SMF_GenerateStatusReport function to write this string to the Report log file. This string could be just as easily written to the Status Log file by using the PGS_SMF_SetDynamicMsg tool.

Note that it is up to the user to specify the alternate action, such as opening the alternate file. **The Toolkit takes no action itself.** That is, the accessing and writing of the action message and the actual action taken are completely independent of each other.

Action labels must not be used as stand-alone messages, i.e., they must never appear explicitly in your code. They can only be tacked on to other messages as in the above example.

Usage of this function is optional.

2.1.2.6 Running the smfcompile utility

Now that preparation of your status message text file is complete, you need to generate files that your program can use -- it does not use the status message text file directly. Do this by executing the **smfcompile** utility.

i) For use in C, the procedure is to run from the Unix command line

```
$PGSBIN/smfcompile -f AVHRR_99.t -r -i
```

This creates two files. *$PGSINC/PGS_PATHFINDER_99.h* is the C include file, and *$PGSMSG/PGS_99* is the **runtime ASCII message file**.

ii) In FORTRAN 77 and FORTRAN 90, run from the Unix command line

```
$PGSBIN/smfcompile -f AVHRR_99.t -f77
```

This creates two files. *$PGSINC/PGS_PATHFINDER_99.f* is the FORTRAN include file, and *$PGSMSG/PGS_99* is the **runtime ASCII message file**.

iii) In Ada, run from the Unix command line

```
$PGSBIN/smfcompile -f AVHRR_99.t -ada
```

This creates two files. *$PGSINC/PGS_PATHFINDER_99.ada* is the Ada package specification file, and *$PGSMSG/PGS_99* is the **runtime ASCII message file**.

You should never modify either one of the two files created by *smfcompile*. The status message text file AVHRR_99.t is the only file you should ever edit.

The runtime ASCII message file is independent of language, while the include or package specification file is language dependent.

Once you have constructed your status message text file, you can modify it. If you only modify the text of the messages, and not the mnemonic labels, then you do not need to recompile your code; you only need to rerun *smfcompile*. This is because the include files (PGS_PATHFINDER.h, .f. or .ada) do not contain the text of the message, only the mnemonic and its internal code. If you do add or change mnemonic labels, then you will need to recompile your code, after rerunning *smfcompile*.

The source code for the smfcompile utility is $PGSSRC/SMF/PGS_SMF_Comp.c .

2.1.3 Log files

Before we get into how to use the messages in your code, an explanation of log files is in order. There are 3 log files generated by the Toolkit: the Status log file, the User log file, and the Report log file. All of these files are opened automatically the first time they are needed. They are identified respectively as LogStatus, LogUser, and LogReport in the default Process Control file $PGS_PC_INFO_FILE ($PGSRUN/PCF.v5), as explained in the Process Control section below. The Toolkit does not delete existing log files, but instead appends new information to them.

In order to use Toolkit log files at the SCF, you must use either PGS_PC_Shell.sh or PGS_PC_InitCom to initialize the Toolkit.
(This is done by the system at the DAAC.)

2.1.3.1 Status log file

The **Status log file** is automatically updated every time either your code or the Toolkit code calls one of the Toolkit functions PGS_SMF_Set*Msg. Thus this file captures all error and status information concerning a program.

Here we explain in detail what you see in the log file, using an example.

```
11:PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_DATA_PRESENT:76807
The data requested is not in the line found.
```

Each entry consists of two lines, followed by a blank line. This example is a warning message generated by a Toolkit function. The first line contains configuration and other information.

```
11:PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_DATA_PRESENT:76807
```

The first number (**1**) is the Production Run ID; the second (**1**) is the Software (version) ID. These parameters are obtained by the Toolkit from the process control file $PGS_PC_INFO_FILE, as explained in the Process Control section below.

```
11:PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_DATA_PRESENT:76807
```

The next entry is the name of the function that set the message, through use of one of the Toolkit functions PGS_SMF_Set*Msg.

```
11:PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_DATA_PRESENT:76807
```

The next entry is the mnemonic label of the message.

```
11:PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_DATA_PRESENT:76807
The final entry on this line is the SMF error code, which is used internally by the Toolkit to identify the
error or status.
```

```
The data requested is not in the line found.
```

The second line is the text of the message. For your messages, this is the message string that you typed into the status message text file AVHRR_99.t, as explained above.

2.1.3.2 User log file

The **User log file** is automatically updated every time your code calls one of the Toolkit functions PGS_SMF_Set*Msg, **and** the message level is of type "_U_" or "_N_". Thus this file consists of the subset of status messages that are of particular interest to you. (No Toolkit functions use messages of these two levels.)

```
11:():PATHFINDER_N_PROCESSING_DONE:813585
SUCCESS: AVHRR processing complete at Mon Sep 19 17:37:47 1994
```

Since this message is of level "_N_", it appears in the User log file (and also the Status log file).

2.1.3.3 Report log file

The **Report log file** is updated each time you make a call to Toolkit function PGS_SMF_GenerateStatusReport. This function takes as input any string, and simply writes it to this file. The messages you generated in AVHRR_99.t are not necessarily used. Thus this file is a way for you to send arbitrary information to a log file. No Toolkit functions call this function, so you are in complete control of what gets written to the Report log file.

2.1.3.4 Where the log files go

The Toolkit writes the log files to directory $PGSHOME/runtime. You can get these files sent to a remote machine through use of either PGS_PC_Shell.sh or PGS_PC_InitCom and PGS_PC_TermCom for more information regarding the sending of files).

2.1.3.5 Log files are not deleted by Toolkit

The Toolkit writes to log files in "append" mode. This means that the log files will remain until you delete them. The log files are designed this way in order to accept input from several executables from a single PGE. When testing at the SCF, you might want to manually delete these files occasionally to save disk space. Alternatively you could delete them in your test script before each run. In the production system, the SDPS will delete the log files between successive executions of a PGE.

2.1.4 Using error/status messages in your code

This section provides pointers to the major functions which you need to use to implement error/status messaging in your code. Only a brief summary is given in this section; the explanations of the individual Toolkit functions, along with detailed examples of usage, appear in the Tool Descriptions section.

2.1.4.1 Writing error/status messages to log files

The simplest thing to do is to save an error message, once your code detects an error. This is done by calling one of the functions PGS_SMF_Set*Msg. The Toolkit automatically writes to the log file the message string corresponding to the mnemonic label which you supply as input. The message is saved in memory to the internal status message buffer for future use. There are 3 tools that perform this function:

Tool PGS_SMF_SetStaticMsg does this for a pre-defined message.

Tool PGS_SMF_SetDynamicMsg does this for dynamic data such as the value of variables at runtime, when used in conjunction with another tool (See section 2.1.4.2 for information on message retrieval tools).

Tool PGS_SMF_SetUNIXMsg does this for error codes returned from Unix system calls.

Since the Toolkit writes these messages to the Status log file automatically, this is all you need to do, if this is all you want.

If you want to write an arbitrary string to a log file at runtime, without benefit of your previously constructed error/status messages, use tool PGS_SMF_GenerateStatusReport . It writes to the Report log file. One use you could make of this method is to write really important messages such as unexpected errors to the Report log file. Such errors are written to the Status log file, but may be hard to separate from the many Toolkit messages in that file. Since you control everything that is written to the Report log file, this will assure that the message gets your attention.

In order for all of the above functions to work, an entry for each log file must appear in the Process Control file $PGS_PC_INFO_FILE. The default version of this file $PGSRUN/PCF.v5 contains these entries already, so if you use this file that is already done for you.

2.1.4.2 Retrieving messages in your code

If for some reason you wish to retrieve the message inside your program, use the PGS_SMF_Get* functions.

PGS_SMF_GetMsg retrieves the message currently in the internal status message buffer, as set previously by a PGS_SMF_Set*Msg function. This message has already been automatically written to the Status log file by the time you do this, so it is not really necessary to ever call this function.

PGS_SMF_GetMsgByCode retrieves a message string given its mnemonic label. It is useful for constructing dynamic messages, as shown in the examples for PGS_SMF_SetDynamicMsg in the Tool Descriptions section.

You can also get the Action part of a given mnemonic label, by calling function PGS_SMF_GetActionByCode . This may be useful if you want to write the action message to the Report log file.

2.1.4.3 Returning error/status codes from your lower-level modules

You may wish to use error/status messages as the return value of your own modules. The advantage to this is that you can then switch on either the mnemonic label code itself, or on its status level, in the module that calls your lower-level function.

To do this, your module must be a function, and it must return a variable of type *PGSt_SMF_status* (C) or *INTEGER* (FORTRAN).

To switch on the status level of a returned value, use the PGS_SMF_Test*Level functions. These include PGS_SMF_TestStatusLevel, which returns the status level given a mnemonic label, and the set of functions PGS_SMF_TestFatalLevel, PGS_SMF_TestErrorLevel, PGS_SMF_TestWarningLevel, PGS_SMF_TestUserInfoLevel, PGS_SMF_TestNoticeLevel, PGS_SMF_TestMessageLevel, and PGS_SMF_TestSuccessLevel, which all return PGS_TRUE or PGS_FALSE depending on whether the input mnemonic label is of that level or not.

2.1.4.4 Sending files to a remote machine

Toolkit function PGS_SMF_SendRuntimeData is used to mark files of your choice for sending to a remote machine.
The actual process of sending both these files and Toolkit log files to the remote machine is handled through use of either PGS_PC_Shell.sh, or PGS_PC_InitCom and PGS_PC_TermCom.

These functions also automatically send email to a user on a remote machine.
Sending of files and email may be disabled by resetting the TransmitFlag (logical 10109) in the Process Control file.

Note that the feature of the Toolkit which allows file and e-mail transmission is indended for SCF use only. In the DAAC environment, these services will be performed through the Data Server subscription mechanism.

2.1.4.5 Miscellaneous functions

PGS_SMF_SetArithmeticTrap accepts the name of your signal handling function, which the system will then use in the event of an arithmetic error, thus avoiding a core dump. **Due to unforeseen implementation difficulties, this tools was never officially released. For details on the problems encountered, please read the signal handling investigation summary.**

PGS_SMF_GetInstrName returns the name of the instrument, given an error/status mnemonic label. PGS_SMF_CreateMsgTag returns a string containing configuration information, for use in stamping your own messages.

2.1.5 How the Toolkit itself uses error/status messages

The Toolkit itself makes extensive use of PGS_SMF_* functions for error checking purposes. Much effort has gone into assuring that the maximum number of possible errors will be trapped, without sacrificing the speed and efficiency of the Toolkit code.

Nearly all Toolkit functions are of type *PGSt_SMF_status* in C, or *INTEGER* in FORTRAN, which means that they return a status or error value that may be checked and acted on using PGS_SMF_* functions.

Toolkit runtime ASCII message files have filenames of the form $PGSMSG/PGS_?, where ? = 1 to 13. They are derived from status message text files with filenames of the form $PGSMSG/PGS_*grp*_?.t, where *grp* = Toolkit group name (SMF, PC, IO, ...) and ? = 1 to 13. The corresponding include files have filenames of the form $PGSINC/PGS_*grp*_?.h (C), $PGSINC/PGS_*grp*_?.f (FORTRAN), and $PGSINC/PGS_*grp*_?.ada (Ada).

The Toolkit bases no action on the severity of error levels; that task is left to the science software. In particular, the Toolkit never returns a fatal error, nor exits a program. In general, returned values from Toolkit functions are either of status levels "_S_", "_W_", or "_E_". The only time the Toolkit itself acts on the status level of a message is when it sends user-generated messages of status level "_N_" or "_U_" to the User log file, as explained in section 2.1.3, Log files.

Switching on the level of error is the province of the PGS_SMF_Test*Level set of tools. These tools are for use in the science software.

Since a message is written to the Status log file every time a PGS_SMF_Set*Msg function is called, many of these messages will be generated by Toolkit functions, in the event of warnings or errors. If a low-level Toolkit function detects an error or warning, it will write a message to the Status log file, then return the appropriate message to its calling function. That function also will write to the log file, if it is unable to handle the error, and return an appropriate error or warning message to its calling function. So a single error or warning can result in several messages in the log file; this enables traceability of the problem. The Status log file is in fact the only source of traceability for Toolkit errors.

There is a special case where warning messages are generated, when in fact there is no anomaly in processing. See the entry for PGS_IO_Gen_Open.

This concludes the "Overview" section of the error/status messaging tools.

2.2.1 PGS_SMF_CreateMsgTag

Short explanation of what it's for: Returns a string containing configuration information, for stamping such things as entries in your Report log file. Currently, configuration consists of Science Software Configuration ID and Production Run ID.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

For the examples, arbitrarily let the Science Software Configuration ID be "V3.0" and the Production Run ID be "SCF22" in the Process Control file (see Notes section).

C example:

```
#include <PGS_SMF.h>
char systemTag[PGSd_SMF_TAG_LENGTH_MAX];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
returnStatus = PGS_SMF_CreateMsgTag(systemTag);
/*
The resultant value of systemTag is "V3.0SCF22"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INTEGER pgs_smf_createmsgtag
      CHARACTER*60 systemtag
      INTEGER returnstatus
C
C Begin example
C
      returnstatus = pgs_smf_createmsgtag(systemtag)
C
C The resultant value of systemtag is "V3.0SCF22"
```

Notes:

Configuration information is read from the SYSTEM RUNTIME PARAMETERS section of the Process Control file $PGS_PC_INFO_FILE (see Process Control tool section), so this file must have been prepared first, and its environment variable set.

DAAC and hardware indentification are being considered as additions to the configuration data, for future deliveries of the Toolkit.

2.2.2 PGS_SMF_GenerateStatusReport

Short explanation of what it's for: Writes an arbitrary string to the Report log file. You may use this function as an alternative to preparing Toolkit SMF style error/status messages.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

Examples that follow show how to write an error message with configuration information to the Report log file. Example given is an error condition returned from Toolkit function PGS_IO_Gen_Open. Although this error message is automatically written to the Status log file by the Toolkit, you might want to write it to the User log file also, in order to more easily identify it as an important message.

Variable "systemTag" has been previously set to value "V3.0SCF22" (see PGS_SMF_CreateMsgTag).

C example:

```
#include <stdio.h>
#include <PGS_SMF.h>
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
char message[1024];
char systemTag[PGSd_SMF_TAG_LENGTH_MAX];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
returnStatus = PGS_IO_Gen_Open(        GOLDEN_BINARY,
                   PGSd_IO_Gen_Read, &processGolden, 1);
/*
   Test whether status level is "_E_" or worse
*/
if( PGS_SMF_TestStatusLevel(returnStatus) >= PGS_SMF_MASK_LEV_E )
{
/*
   If error, prepare msg, write to Report log file
*/
   sprintf( message, "%s : Error opening golden binary file\n",
                        systemTag );
   returnStatus = PGS_SMF_GenerateStatusReport(message);
/*
   The string "V3.0SCF22 :  Error opening golden binary file"
   is written to the Report log file
*/
}
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_teststatuslevel
      INTEGER pgs_smf_generatestatusreport
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      CHARACTER*1024 message
      CHARACTER*60 systemtag
      INTEGER returnstatus
C
C Begin example
C
C Try to open a file
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
     .                        PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
C
C Test whether status level is "_E_" or worse
      IF ( pgs_smf_teststatuslevel(returnstatus) .GE.
     .                        PGS_SMF_MASK_LEV_E ) THEN
C
C If error, prepare msg, write to Report log file
      message = systemtag // 'Error opening golden binary file'
      returnstatus =
pgs_smf_generatestatusreport(message)

C
C The string "V3.0SCF22 :  Error opening golden binary file"
C  is written to the Report log file
C
      END IF
```

Notes:

Message passed to this function may be of arbitrary length.

Report log file name is read internally by Toolkit functions from the SUPPORT OUTPUT section of the Process Control file $PGS_PC_INFO_FILE (see the Process Control tool section), the first time PGS_SMF_GenerateStatusReport is called. This file must have been prepared first, and its environment variable set. Normally you would not change this entry from the Process Control file template supplied with the Toolkit ($PGSRUN/PCF.v5).

The golden binary file of the example must have an entry in the Process Control file, i.e., integer 401 must be associated with a reference (physical filename) in that file.

Log files may be sent to a remote machine through use of either PGS_PC_Shell.sh, or PGS_PC_InitCom and PGS_PC_TermCom.

2.2.3 PGS_SMF_GetActionByCode

Short explanation of what it's for: Retrieves action string portion of an error/status action definition, given its mnemonic label code. You might use this to write the action string explicitly to the Report log file, for example.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

The examples assume that the following entries exist in the status message text file AVHRR_99.t :

```
PATHFINDER_A_ALT_FILE_USED      Alternate file used
PATHFINDER_W_OZONE_FILE_MISSING        Ozone file not found
                                  ::PATHFINDER_A_ALT_FILE_USED
```

C example:

```
#include <PGS_SMF.h>
char actionString[PGSd_SMF_MAX_ACT_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
returnStatus = YourLowerLevelModule( arg1, arg2, ... );
if( returnStatus == PATHFINDER_W_OZONE_FILE_MISSING)
{
   returnStatus = PGS_SMF_GetActionByCode(
                      PATHFINDER_W_OZONE_FILE_MISSING, actionString );
/*
   actionString now contains the string "Alternate file used"
*/
   returnStatus = PGS_SMF_GenerateStatusReport(actionString);
/*
   The string "Alternate file used" is written to the
   Report log file
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INTEGER pgs_smf_getactionbycode
      INTEGER yourlowerlevelmodule
      INTEGER pgs_smf_generatestatusreport
      CHARACTER*240 actionstring
      INTEGER returnstatus
C
C Begin example
C
      returnstatus = yourlowerlevelmodule( arg1, arg2, ... )
      IF( returnstatus .EQ. PATHFINDER_W_OZONE_FILE_MISSING) THEN
         returnstatus = pgs_smf_getactionbycode(
     .             PATHFINDER_W_OZONE_FILE_MISSING, actionstring )
C
C  actionstring now contains the string 'Alternate file used'
C
        returnstatus = pgs_smf_generatestatusreport(actionString)
C
C  The string 'Alternate file used' is written to the
C  Report log file
C
      END IF
```

Notes:

Messages must have been prepared in a status message text file first, and run through the smfcompile utility.

2.2.4 PGS_SMF_GetInstrName

Short explanation of what it's for: Returns the name of the instrument corresponding to a given mnemonic label code. This may be useful for determining which instrument generated an error/status message, in the case of code integrated between more than one instrument.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

Examples assume the first four lines of the status message text file AVHRR_99.t appear as follows:

```
%INSTR        = AVHRR
%LABEL        = PATHFINDER
%SEED         = 99
PATHFINDER_F_OPEN_BINARY_FILE FATAL_ERROR...opening binary file
```

C example:

```
#include <PGS_SMF.h>
char instr[PGS_SMF_MAX_INSTR_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
returnStatus = PGS_SMF_GetInstrName(
                    PATHFINDER_F_OPEN_BINARY_FILE, instr );
/*
   instr now contains the string "AVHRR"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INTEGER pgs_smf_getinstrname
      CHARACTER*10 instr
      INTEGER returnstatus
C
C Begin example
C
returnstatus = pgs_smf_getinstrname(
                       PATHFINDER_F_OPEN_BINARY_FILE, instr )
C
C  instr now contains the string 'AVHRR'
C
```

Notes:

Messages must have been prepared in a status message text file first, and run through the smfcompile utility.

2.2.5 PGS_SMF_GetMsg

Short explanation of what it's for: Retrieves the current error/status message from the message buffer. It is normally not necessary to call this function in production processing.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

Examples show how to use this function to print to the screen during development. Note that the recommended error handling for production is very different; see PGS_SMF_SetStaticMsg.

The examples contain two levels: (1) your main module, and (2) your lower level module, which attempts to open a file using PGS_IO_Gen_Open (C) or PGS_IO_Gen_OpenF (FORTRAN).

The following entry is assumed to appear in the status message text file AVHRR_99.t:

```
PATHFINDER_F_OPEN_BINARY_FILE FATAL_ERROR...opening binary file
```

The corresponding entry then must appear in the runtime ASCII message file PGS_99:

```
815107,PATHFINDER_F_OPEN_BINARY_FILE,NULL,FATAL_ERROR...error opening
binary file
```

C example:

```
#include <PGS_SMF.h>
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_SMF_code code;
char mnemonic[PGS_SMF_MAX_MNEMONIC_SIZE];
char message[PGS_SMF_MAX_MSGBUF_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Call low-level module, whose definition is given below
*/
returnStatus = LowLevelModule( &processGolden );
if( returnStatus != PGS_S_SUCCESS )
{
/*
   Get error message from buffer, print to screen
*/
   PGS_SMF_GetMsg( &code, mnemonic, message );
   printf("LowLevelModule: %s\n", message );
/*
   Values of the variables returned by PGS_SMF_GetMsg are
        code: 815107
        mnemonic: "PATHFINDER_F_OPEN_BINARY_FILE"
        message: "FATAL_ERROR...opening binary file"
   The string "LowLevelModule: FATAL_ERROR...opening binary file"
   is printed to the screen.
*/
}
/*
End main module
*/
/*
Low level module definition
*/
PGSt_SMF_status LowLevelModule(
                PGSt_IO_Gen_FileHandle **processGoldenPtr )
{
/*
   Try to open a file
*/
returnStatus = PGS_IO_Gen_Open(        GOLDEN_BINARY,
                 PGSd_IO_Gen_Read, processGoldenPtr, 1);
/*
   Test whether status level is "_E_" or worse
*/
if( PGS_SMF_TestStatusLevel(returnStatus) >= PGS_SMF_MASK_LEV_E )
{
   PGS_SMF_SetStaticMsg( PATHFINDER_F_OPEN_BINARY_FILE, "" );
   return(PATHFINDER_F_OPEN_BINARY_FILE);
}
return(PGS_S_SUCCESS);
}
/*
End low level module definition
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER lowlevelmodule
      INTEGER pgs_smf_getmsg
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      INTEGER code
      CHARACTER*32 mnemonic
      CHARACTER*480 message
      INTEGER returnstatus
C
C Begin example
C
C Call low-level module, whose definition is given below
C
      returnstatus = lowlevelmodule( processgolden )
      IF( returnstatus != PGS_S_SUCCESS ) THEN
C
C  Get error message from buffer, print to screen
C
         call pgs_smf_getmsg( code, mnemonic, message )
         PRINT *, 'LowLevelModule: ', message
C
C  Values of the variables returned by PGS_SMF_GetMsg are
C        code: 815107
C        mnemonic: 'PATHFINDER_F_OPEN_BINARY_FILE'
C        message: 'FATAL_ERROR...opening binary file'
C  The string 'LowLevelModule: FATAL_ERROR...opening binary file'
C  is printed to the screen.
C
      END IF
C
C End main module
C
C Low level module definition
C
      INTEGER FUNCTION LowLevelModule( processgolden )
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_teststatuslevel
      INTEGER processgolden
C
C  Try to open a file
C
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
     .              PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
C
C  Test whether status level is "_E_" or worse
C
      IF ( pgs_smf_teststatuslevel(returnstatus) .GE.
     .          PGS_SMF_MASK_LEV_E ) THEN
         pgs_smf_setstaticmsg( PATHFINDER_F_OPEN_BINARY_FILE, '' )
         RETURN(PATHFINDER_F_OPEN_BINARY_FILE)
      END IF
      RETURN(PGS_S_SUCCESS);
      END

C
C End low level module definition
C
```

Notes:

Message to be retrieved must have been previously written to the message buffer by one of the PGS_SMF_Set*Msg functions, e.g., PGS_SMF_SetStaticMsg .

This function retrieves what is currently in the message buffer.

The reason it is normally not necessary to call this function in production processing is that any message in the message buffer is automatically written to the Status log file internally by the Toolkit.

Messages must have been prepared in a status message text file first, and run through the smfcompile utility.

The golden binary file of the example must have an entry on the Process Control file, i.e., integer 401 must be associated with a reference (physical filename) in that file.

In the example, note that the values returned by the PGS_SMF_GetMsg call are independent of the return value of the LowLevelModule function, though they are associated with each other.

2.2.6 PGS_SMF_GetMsgByCode

Short explanation of what it's for: Returns the message definition string corresponding to a given mnemonic label code. Primary use is to enable creation of dynamic messages.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

This example first shows how to retrieve a message definition string for a given mnemonic label code using PGS_SMF_GetMsgByCode, then shows how the result can be used to create a dynamic message.

The following entry is assumed to appear in the status message text file, AVHRR_99.t:

C example:

```
PATHFINDER_F_OPEN_ANC_FILE                FATAL_ERROR...%s
```

FORTRAN example:

```
PATHFINDER_F_OPEN_ANC_FILE                ('FATAL_ERROR...', A)
```

C example:

```
#include <PGS_IO.h>
#include <PGS_SMF.h>
#include <PGS_AVHRR_99.h>

#define GOLDEN_BINARY 401
#define SUCCESS 0       /* GSFC AVHRR/Pathfinder error handling */
#define FATAL_ERROR -1  /* GSFC AVHRR/Pathfinder error handling */
char descrip[80];
char message[PGS_SMF_MAX_MSGBUF_SIZE];
char buf[PGS_SMF_MAX_MSGBUF_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
strcpy( descrip, "opening golden binary file" );
if (PGS_IO_Gen_Open( GOLDEN_BINARY, PGSd_IO_Gen_Read,
                     processGoldenPtr, 1)
       != PGS_S_SUCCESS) goto EXCEPTION;
.
.
.
return (SUCCESS);
/*
   Exception block
*/
EXCEPTION:
/*
   First retrieve the static message string into 'message'
*/
   PGS_SMF_GetMsgByCode( PATHFINDER_F_OPEN_ANC_FILE, message );
/*
   message now contains the string "FATAL_ERROR...%s"
*/
   sprintf( buf, message, descrip );
/*
   buf now contains the string
      "FATAL_ERROR...opening golden binary file"
*/
   PGS_SMF_SetDynamicMsg( PATHFINDER_F_OPEN_ANC_FILE, buf, "" );
/*
   Message buffer now contains the buf string, and it has been
      automatically written to the Status log file
*/
   return(FATAL_ERROR);
}
```

FORTRAN example:

```
      include 'PGS_IO.f'
      include 'PGS_SMF.f'
      include 'PGS_AVHRR_99.f'

      character*80 descrip
      character*240 message
      character*240 buf
      character*20  func_name

      integer golden_binary
      integer success        ! GSFC AVHRR/Pathfinder error handling
      integer fatal_error    ! GSFC AVHRR/Pathfinder error handling
      integer golden_un
      integer version
      integer returnstatus

!  Begin example

!  Initialize variables

      func_name = 'avhrr_func()'  ! name of this function
      success = 0                 ! success return value
      fatal_error = -1            ! error return value
      golden_binaray = 401        ! file logical ID in PCF
      version = 1

!  Try to open a file

      descrip = 'opening golden binary file'
      returnstatus = pgs_io_gen_openf( golden_binary,
     >                                 pgsd_io_gen_rseqfrm, 0,
     >                                 golden_un, version )

      if ( returnstatus .ne. pgs_s_success ) goto 999
.
      avhrr_func = success
      return

!  Exception block

 999  continue

!  First retrieve the static message string into 'message'

   pgs_smf_getmsgbycode( PATHFINDER_F_OPEN_ANC_FILE, message )

!  message now contains the string '('FATAL_ERROR...', A)'

      write(buf, ref=message) descrip

!  buf now contains the string:
!  'FATAL_ERROR...opening golden binary file

      pgs_smf_setdynamicmsg(PATHFINDER_F_OPEN_ANC_FILE, buf, func_name)

!  Message buffer now contains the buf string, and it has been
!  automatically written to the Status log file

      avhrr_func = fatal_error
      return
```
Notes:

The golden binary file of the example must have an entry on the Process Control file, i.e., integer 401 must be associated with a reference (physical filename) in that file.

Messages must have been prepared in a status message text file first, and run through the smfcompile utility.

2.2.7 PGS_SMF_SendRuntimeData

Short explanation of what it's for: Mark output files for sending to a designated machine. Email is also sent to designated recipient(s).

This function is in file: $PGSSRC/SMF/PGS_SMF_SendRuntimeData.c

Examples:

The examples assume the following exists in the Process Control File (PCF):

```
?   PRODUCT INPUT FILES
#
201|87002002709.no9_gac||||gac_attributes|1
399|test10.hdf|||||3
399|test23.hdf|||||2
399|test06.hdf|||||1
```

C example:

```
#include <PGS_SMF.h>
#define GAC_FILE   201
#define HDF_INFILE 399
PGSt_integer sendFile[3];
PGSt_integer version[3];
PGSt_integer numFiles;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
sendFile[0] = GAC_FILE;
version[0] = 1;

sendFile[1] = HDF_INFILE;
version[1] = 1;

sendFile[2] = HDF_INFILE;
version[2] = 2;

numFiles = 3;

returnStatus = PGS_SMF_SendRuntimeData(
                      numFiles, sendFile, version );

/* Files 87002002709.no9_gac, test10.hdf, and test23.hdf
   in default directory $PGS_PRODUCT_INPUT are now marked for
   sending to a remote machine. */
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INTEGER pgs_smf_sendruntimedata
      INTEGER GAC_FILE
      PARAMETER (GAC_FILE=201)
      INTEGER HDF_INFILE
      PARAMETER (HDF_INFILE=399)
      INTEGER sendfile(3)
      INTEGER version(3)
      INTEGER numfiles
      INTEGER returnstatus
C
C Begin example
C
      sendfile(1) = GAC_FILE
      version(1) = 1

      sendfile(2) = HDF_INFILE
      version(2) = 1

      sendfile(3) = HDF_INFILE
      version(3) = 2

      numfiles = 2
      returnstatus = pgs_smf_sendruntimedata(
     .                   numfiles, sendfile, version )

C Files 87002002709.no9_gac, test10.hdf, and test23.hdf
C in default directory $PGS_PRODUCT_INPUT are now marked for
C sending to a remote machine.
```

Notes:

This function should only be called once in your program. Only the last call of this function is recognized by the Toolkit. This is to minimize overhead in file transfers.

The mechanism for doing the file transfers is currently Unix function ftp. This may change in the future, e.g., to use DCE or its equivalent, but the calling sequence will not change. In the production environment, files will be placed on an intermediate machine; then you can retrieve them at your convenience.

In the examples, of the 3 files with logical ID 399, files *test10.hdf* and *test23.hdf* are sent because they are listed #1 and #2 in order in the PCF. For more information about version numbers vs. sequence numbers (sequence numbers are the ones listed in the last field of the PCF entries), see "sec. 4.1.2.2, Constructing your Process Control file, PRODUCT INPUT, Field 7, Sequence number".

In order for this function to work at the SCF, you must be using either PGS_PC_Shell.sh or Edit line 10106 to change "sandcrab" to the machine to which you want the files sent. In the production environment, this will be an intermediate machine, from which you will later retrieve the file.

Edit line 10107 to change "/usr/kwan/test/PC/data" to the fully qualified directory name to which you want the files written on that machine. The directory must exist, and have at least Unix user permission "w".

Edit line 10108 to change "kwan@eos.hitc.com" to the email address at which you want email notification that the files have been sent.

In your home directory $HOME on the machine on which you are executing your code at the SCF, the file ".netrc" must exist, with Unix permissions "-rw-------". This file must contain the line

machine sandcrab login kwan password kwan_password

where "sandcrab" is the machine of line 10106, "kwan" is a valid user of that machine, and "kwan_password" is his/her password.

Due to local security policy, the use of this mechanism of using '.netrc' files may be changed. The production environment may use a more secure mechanism.

2.2.8 PGS_SMF_SetArithmeticTrap

Short explanation of what it's for: Catching arithmetic errors in your program, in order to avoid core dumps.

Notes:

This tool was intended to be delivered with TK4. However, implementation has been problematic. At this writing delivery of the tool is uncertain. For details on the problems encountered, please read the signal handling investigation summary.

2.2.9 PGS_SMF_SetDynamicMsg

Short explanation of what it's for: Saves a runtime-defined error/status message to the message buffer. Every time this function is called, it writes an entry to the Status log file.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

There are at least two different ways to use this function. Example 1 shows C programmers how to construct dynamic messages through use of PGS_SMF_GetMsgByCode; Example 2 shows how to do it directly, in both C and FORTRAN.

Example 1: Using PGS_SMF_GetMsgByCode

This example first shows how to retrieve a message definition string for a given mnemonic label code using PGS_SMF_GetMsgByCode, then shows how the result can be used to create a dynamic message.

The following entry is assumed to appear in the status message text file , AVHRR_99.t:

```
PATHFINDER_F_OPEN_ANC_FILE    FATAL_ERROR...%s
```

The corresponding entry then must appear in the runtime ASCII message file PGS_99:

```
815104,PATHFINDER_F_OPEN_ANC_FILE,NULL,FATAL_ERROR...%s
```

C example 1:

```
#include <PGS_IO.h>
#include <PGS_SMF.h>
#define GOLDEN_BINARY 401
#define SUCCESS 0       /* GSFC AVHRR/Pathfinder error handling */
#define FATAL_ERROR -1  /* GSFC AVHRR/Pathfinder error handling */
char descrip[80];
char message[PGS_SMF_MAX_MSGBUF_SIZE];
char buf[PGS_SMF_MAX_MSGBUF_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
strcpy( descrip, "opening golden binary file" );
if (PGS_IO_Gen_Open( GOLDEN_BINARY, PGSd_IO_Gen_Read,
                         processGoldenPtr, 1)
         != PGS_S_SUCCESS) goto EXCEPTION;
.
.
.
return (SUCCESS);
/*
   Exception block
*/
EXCEPTION:
/*
   First retrieve the static message string into 'message'
*/
   returnStatus = PGS_SMF_GetMsgByCode(
                      PATHFINDER_F_OPEN_ANC_FILE, message );
/*
   message now contains the string "FATAL_ERROR...%s"
*/
   sprintf( buf, message, descrip );
/*
   buf now contains the string
      "FATAL_ERROR...opening golden binary file"
*/
   returnStatus = PGS_SMF_SetDynamicMsg(
                         PATHFINDER_F_OPEN_ANC_FILE, buf, "" );
/*
   Message buffer now contains the buf string, and the following
   entry appears in the Status log file:
11::PATHFINDER_F_OPEN_ANC_FILE:815104
FATAL_ERROR...opening golden binary file
*/
   return(FATAL_ERROR);
}
```

FORTRAN example 1:

Work is in progress on a FORTRAN function that is essentially a wrapper on C function "sprintf", which will enable the creation of dynamic messages in FORTRAN, similar to that given in the C example above. This method cannot be applied in FORTRAN until the "sprintf" wrapper is available. For now, FORTRAN users can only use the method of Example 2 below to construct dynamic messages.

Example 2: Direct Method

This example shows how to construct dynamic messages directly.

C example 2:

```
#include <PGS_IO.h>
#include <PGS_SMF.h>
#define GOLDEN_BINARY 401
#define SUCCESS 0       /* GSFC AVHRR/Pathfinder error handling */
#define FATAL_ERROR -1  /* GSFC AVHRR/Pathfinder error handling */
char descrip[80];
char message[PGS_SMF_MAX_MSGBUF_SIZE];
char buf[PGS_SMF_MAX_MSGBUF_SIZE];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
strcpy( descrip, "opening golden binary file" );
if (PGS_IO_Gen_Open( GOLDEN_BINARY, PGSd_IO_Gen_Read,
                      processGoldenPtr, 1)
         != PGS_S_SUCCESS) goto EXCEPTION;
.
.
.
return (SUCCESS);
/*
   Exception block
*/
EXCEPTION:
/*
   Construct error string manually
*/
   strcpy( buf, "FATAL_ERROR..." );
   strcat( buf, descrip );
/*
   buf now contains the string
      "FATAL_ERROR...opening golden binary file"
*/
   returnStatus = PGS_SMF_SetDynamicMsg(
                       PATHFINDER_F_OPEN_ANC_FILE, buf, "" );
/*
   Message buffer now contains the buf string, and the following
   entry appears in the Status log file:
11::PATHFINDER_F_OPEN_ANC_FILE:815104
FATAL_ERROR...opening golden binary file
*/
   return(FATAL_ERROR);
}
```

FORTRAN example 2:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_setdynamicmsg
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      CHARACTER*32 mnemonic
      CHARACTER*480 buf
      CHARACTER*480 message
      INTEGER returnstatus
C

C Begin example
C
C  Try to open a file
C
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
     .                        PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
      IF (returnstatus .NE. PGS_S_SUCCESS ) GO TO 999
.
.
.
      RETURN (SUCCESS)
C
C  Exception block
C
999   CONTINUE
C
C  Construct error string manually
C
      buf = 'FATAL_ERROR...' // descrip
C
C  buf now contains the string
C     'FATAL_ERROR...opening golden binary file'
C
      returnstatus =  pgs_smf_setdynamicmsg(
     .             PATHFINDER_F_OPEN_ANC_FILE, buf, '' )
C
C  Message buffer now contains the buf string, and the following
C  entry appears in the Status log file:
C 11::PATHFINDER_F_OPEN_ANC_FILE:815104
C FATAL_ERROR...opening golden binary file
C
      RETURN(FATAL_ERROR)
      END
```

Notes:

The value of the second argument of PGS_SMF_SetDynamicMsg *buf* is what is stored in the message buffer. The only effect of the first argument (the mnemonic label code) is that the mnemonic label code and string are written to the Status log file. The message that is permanently associated with the mnemonic label for this PGE run, as defined in the runtime ASCII message file PATHFINDER_99, does not change.

Both this method of using PGS_SMF_SetDynamicMsg (Example 2), and tool PGS_SMF_GenerateStatusReport, write a user-defined string to a log file. The differences between them are (1) the two methods write to different log files, (2) the log file entry from PGS_SMF_SetDynamicMsg contains an extra line with mnemonic label string, code, etc., and (3) PGS_SMF_SetDynamicMsg saves its message to the message buffer, for optional later retrieval.

Messages must have been prepared in a status message text file first, and run through the smfcompile utility.

2.2.10 PGS_SMF_SetStaticMsg

Short explanation of what it's for: Saves a pre-defined error/status message to the message buffer. This is the primary mechanism you use to handle errors that have static messages associated with them. Every time this function is called, it writes an entry to the Status log file.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

The example attempts to open a file using PGS_IO_Gen_Open.

The following entry is assumed to appear in the status message text file AVHRR_99.t:

```
PATHFINDER_F_OPEN_BINARY_FILE   FATAL_ERROR...opening binary file
```

The corresponding entry then must appear in the runtime ASCII message file PGS_99:

```
815107,PATHFINDER_F_OPEN_BINARY_FILE,NULL,FATAL_ERROR...error opening
binary file
```

C example:

#include <PGS_SMF.h>

```
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
returnStatus = PGS_IO_Gen_Open(        GOLDEN_BINARY,
                    PGSd_IO_Gen_Read, &processGolden, 1);
if( returnStatus != PGS_S_SUCCESS )
{
   PGS_SMF_SetStaticMsg( PATHFINDER_F_OPEN_BINARY_FILE,
                              "YourModuleNameHere" );
/*
   The following entry appears in the Status log file:
11:YourModuleNameHere:PATHFINDER_F_OPEN_BINARY_FILE:815107
FATAL_ERROR...error opening binary file
*/
   return( PATHFINDER_F_OPEN_BINARY_FILE );
}
return( PGS_S_SUCCESS );
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_setstaticmsg
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      INTEGER returnstatus
C
C Begin example
C
C  Try to open a file
C
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
                    PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
      IF ( returnstatus .NE. PGS_S_SUCCESS ) THEN
         pgs_smf_setstaticmsg( PATHFINDER_F_OPEN_BINARY_FILE,
     .                     'yourmodulenamehere' )
         RETURN(PATHFINDER_F_OPEN_BINARY_FILE)
C
C   The following entry appears in the Status log file:
C 11:yourmodulenamehere:PATHFINDER_F_OPEN_BINARY_FILE:815107
C FATAL_ERROR...error opening binary file
C
      END IF
      RETURN(PGS_S_SUCCESS);
```

Notes:

Ordinarily, you never need to retrieve the message saved to the buffer by this function, since it is written to the Status log file. If you want to retrieve it, e.g., for printing to the screen during SCF development, use PGS_SMF_GetMsg. For a detailed explanation of what is written to the Status log file, see section 3.1.3, "Log files".

2.2.11 PGS_SMF_SetUNIXMsg

Short explanation of what it's for: Saves a Unix-defined error/status message to the message buffer. Every time this function is called, it writes an entry to the Status log file.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Examples:

This hypothetical example shows how to trap the error if an unknown I/O error occurs while your program is using a native language I/O function to read from a file.

C example:

```
#include <PGS_SMF.h>
#define GOODESX 5004
#define GOODESY 2168
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_SMF_status returnStatus;
int jc,ret;
short shortvals[GOODESX];
/*
Begin example
*/
for (jc=0;jc<GOODESY;jc++)
{
/*
   Read from a previously opened file
*/
   ret=fread(shortvals,sizeof(short),GOODESX,processGolden);
/*
If error detected in "fread" or EOF, go to exception block
*/
   if (        (ret!=GOODESX) ||
                (feof(processGolden)!=0)||
                (ferror(processGolden)!=0)        )
      goto EXCEPTION;
.
.
.
}
.
.
.
return(PGS_S_SUCCESS);
/*
Exception block
*/
EXCEPTION:
/*
Save Unix error string in buffer, write to log file
Add string indicating error occurred while reading land/sea flags
Return error code for processing in calling module
*/
   returnStatus = PGS_SMF_SetUNIXMsg( errno,
      " -- error reading land/sea flags", "YourModuleNameHere" );
   return(PGS_E_UNIX);
/*
   The following entry appears in the Status log file:
11:YourModuleNameHere:PGS_E_UNIX:1798
UNIX: errno=5, I/O error -- error reading land/sea flags
("I/O error" is the Unix error string associated with
errno=5)
*/
```

FORTRAN example:

This example shows how to check for errors in obtaining the time of the system clock, i.e., the number of seconds elapsed since Jan. 1, 1970. (The example used is different from the C example because of the considerations explained in the Notes section below.)

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.h'
      INTEGER pgs_smf_setunixmsg
      INTEGER itime
      INTEGER ierror
C
C Begin example
C
C Call POSIX FORTRAN subroutine to get system time
C
      CALL PXFTIME( itime, ierror )
      IF( ierror .NE. 0) THEN
C
C Save Unix error string in buffer, write to log file
C
         pgs_smf_setunixmsg( ierror, '',
     .                       'YourModuleNameHere' );
C
C   The following entry appears in the Status log file:
C 11:YourModuleNameHere:PGS_E_UNIX:1798
C UNIX: errno=999, No system time
C [Note that this errno is fabricated]
C
      END IF
```

Notes:

FORTRAN users must have compiled their code with a FORTRAN POSIX compiler in order to use PGS_SMF_SetUNIXMessage; in addition, this function only traps errors occurring through use of FORTRAN POSIX functions. This is because the *ierror* parameter is only returned by FORTRAN POSIX functions. For reading and writing files, for example, this means that in order to use this tool, you must use the POSIX FORTRAN functions PXFREAD and PXFWRITE, and not the ANSI FORTRAN functions READ and WRITE. Unfortunately these POSIX functions only are for stream, or CHARACTER, I/O in FORTRAN, and hence are of limited use in the ECS environment; in addition, we are aware that few SCFs are likely to have FORTRAN POSIX compilers at this point. We recommend that in practice you stick with your current FORTRAN error handling for I/O functions. The fact that this function is of limited use to FORTRAN programmers is essentially a consequence of the fact that FORTRAN makes limited use of system calls.

Further explanation of the Status log file entry of the example is in order.

```
11:YourModuleNameHere:PGS_M_UNIX:1798
UNIX: errno=5, I/O error -- error reading land/sea flags
```

The first line of an error written by a call to PGS_SMF_SetUNIXMsg is always the same, except for that name of the module YourModuleNameHere (assuming you provided this in the call to the function). The second line gives the Unix *errno*, as defined in the system include file errno.h (usually in /usr/include or /usr/include/sys). The string "I/O error" is also from that system file. The example also shows how to add your own string, in this case " -- error reading land/sea flags", to the Unix system error string. The additional string is optional; set it to NULL (C) or '' (FORTRAN) if it is not needed.

2.2.12 PGS_SMF_Test*Level

Short explanation of what it's for: To test the status level severity of a returned value of a Toolkit function or your own lower level module.

This function is in file: $PGSSRC/SMF/PGS_SMF.c

Special Note: This section covers a whole group of functions, all of whose names are of the form PGS_SMF_Test*Level. Here "*" represents either Status, Fatal, Error, Warning, Message, UserInfo, or Success. For further explanation see the explanation of status level in section 3.1.2.3, "Constructing the status message text file".

There are two sub-groups of PGS_SMF_Test*Level functions: one, PGS_SMF_TestStatusLevel, and two, all the rest. PGS_SMF_TestErrorLevel is taken as a typical example of group two; the other functions all behave similarly.

Examples:

The first example is of the first sub-group; it shows how to branch if a Toolkit function returns an error of level "_E_" or "_F_".

The second example is of the second sub-group; it shows how to branch if a Toolkit function returns an error of level "_E_".

In both examples, the science software is interpreting a Toolkit error return code of level "_E_" to be a fatal error. (The Toolkit never returns an error code of level "_F_".)

The following entry is assumed to appear in the status message text file AVHRR_99.t:

```
PATHFINDER_F_OPEN_BINARY_FILE   FATAL_ERROR...opening binary file
```

Example 1: PGS_SMF_TestStatusLevel

This function takes as input a mnemonic label code, and returns one of the following values, corresponding to the status level of the mnemonic:

```
    PGS_SMF_MASK_LEV_S
    PGS_SMF_MASK_LEV_M
    PGS_SMF_MASK_LEV_U
    PGS_SMF_MASK_LEV_N
    PGS_SMF_MASK_LEV_W
    PGS_SMF_MASK_LEV_E
    PGS_SMF_MASK_LEV_F
```

The values of these mnemonics (which are hexadecimals) increase from top to bottom in the above list. That is, e.g., PGS_SMF_MASK_LEV_F is greater than PGS_SMF_MASK_LEV_E.

C example 1:

```
#include <PGS_SMF.h>
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
returnStatus = PGS_IO_Gen_Open( GOLDEN_BINARY, PGSd_IO_Gen_Read,
                              &processGolden, 1 );
/*
   Test whether status level is "_E_" or worse
*/
if( PGS_SMF_TestStatusLevel(returnStatus) >= PGS_SMF_MASK_LEV_E )
{
   PGS_SMF_SetStaticMsg(PATHFINDER_F_OPEN_BINARY_FILE,
                             "YourModuleNameHere");
/*
   The following entry appears in the Status log file:
11:YourModuleNameHere:PATHFINDER_F_OPEN_BINARY_FILE:815107
FATAL_ERROR...error opening binary file
*/
}
```

FORTRAN example 1:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_teststatuslevel
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      INTEGER returnstatus
C
C Begin example
C
C Try to open a file
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
     .               PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
C
C Test whether status level is "_E_" or worse ("_F_")
      IF ( pgs_smf_teststatuslevel(returnstatus) .GE.
     .              PGS_SMF_MASK_LEV_E ) THEN
C
         pgs_smf_setstaticmsg( PATHFINDER_F_OPEN_BINARY_FILE,
     .             'yourmodulenamehere' )
C
C   The following entry appears in the Status log file:
C 11:yourmodulenamehere:PATHFINDER_F_OPEN_BINARY_FILE:815107
C FATAL_ERROR...error opening binary file
C
      END IF
```

Example 2: PGS_SMF_TestErrorLevel

This example also be applies to the Fatal, Warning, Message, UserInfo, and Success functions of class PGS_SMF_Test*Level. These functions return either PGS_TRUE or PGS_FALSE, depending on whether the mnemonic label error code is of the given status level or not. For example, PGS_SMF_TestFatalLevel(PATHFINDER_F_OPEN_BINARY_FILE) returns PGS_TRUE, while the rest of the functions of this sub-group return PGS_FALSE for this argument.

C example 2:

```
#include <PGS_SMF.h>
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Try to open a file
*/
returnStatus = PGS_IO_Gen_Open(        GOLDEN_BINARY,
                     PGSd_IO_Gen_Read, &processGolden, 1);
/*
   Test whether status level is "_E_"
*/
if( PGS_SMF_TestErrorLevel(returnStatus) == PGS_TRUE )
{
   PGS_SMF_SetStaticMsg(PATHFINDER_F_OPEN_BINARY_FILE,
                      "YourModuleNameHere");
/*
   The following entry appears in the Status log file:
11:YourModuleNameHere:PATHFINDER_F_OPEN_BINARY_FILE:815107
FATAL_ERROR...error opening binary file
*/
}
```

FORTRAN example 2:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_openf
      INTEGER pgs_smf_setstaticmsg
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER processgolden
      INTEGER returnstatus
C
C Begin example
C
C Try to open a file
      returnstatus = pgs_io_gen_openf(        GOLDEN_BINARY,
     .                 PGSd_IO_Gen_RSeqUnf, 0, processgolden, 1)
C
C Test whether status level is "_E_"
      IF ( pgs_smf_testerrorlevel(returnstatus) .EQ.
     .              PGS_TRUE ) THEN
C
         pgs_smf_setstaticmsg( PATHFINDER_F_OPEN_BINARY_FILE,
     .                         'yourmodulenamehere' )
C
C   The following entry appears in the Status log file:
C 11:yourmodulenamehere:PATHFINDER_F_OPEN_BINARY_FILE:815107
C FATAL_ERROR...error opening binary file
C
      END IF
```

Notes:

Although the examples given are of Toolkit functions returning SMF error codes and then being tested, you can also use these functions to test your own modules, providing they are set up to return SMF error codes.

Please note that the Toolkit never returns a message of level "_F_" (fatal). Therefore the two examples given are actually equivalent ways of doing the same thing.

In reality, if your code detects a fatal error, you will want to branch use the exit() function (C) or STOP statement (FORTRAN). Codes to use for the argument of this function are not yet defined.

3. Process Control (PC) Tools

3.1 Overview

3.1.1 Introduction

The next highest level of the Toolkit above the SMF tools includes the Process Control (PC) tools. Their purpose is to provide a direct interface between the science software and the rest of the SDPS, including accessing file attributes (data about files), physical filenames (for use by HDF functions), and other functions. These tools are used internally by many Toolkit functions, such as Generic I/O, Ancillary Access, and other tools.

There are two sets of PC tools: the **Command** tools, which are callable from Unix shell scripts, and the **API** tools, callable from C and Fortran. Much of the functionality is duplicated between these two groups; many of the Command tools are simple wrappers on the C code of the API tools, with some exceptions.
For more information about the Command tools see below.
Note: Most of the information in this overview applies to both Command tools and API tools; in particular, both read from the same Process Control File.

The Process Control File (PCF) is central to the PC tools. At the SCF, you construct a PCF using a text editor, one for each PGE. These PCFs are part of the delivery of your software to the DAAC. Your software will access files by logical identifiers (essentially integers, defined by mnemonics). The PCF maps these logical identifiers to physical references (currently physical file names and directories). Each logical identifier corresponds to one or more physical references, or versions. At the SCF, you can use any physical reference you like. In the production environment, the physical reference is supplied by the DAAC. Details are given below.

In this overview section, we walk you through the procedure of constructing your own Process Control File step-by-step, then explain the workings of the pccheck utility, which checks the format of this file. The PCF is read by most of the PC tools (directly or indirectly), and is the current mechanism by which the Toolkit interfaces with the rest of the SDPS. The mechanism may change in the future, but the interface to your code will not.

3.1.2 Constructing your Process Control file

This section explains how to customize a Process Control File for use in your code.

A default Process Control File (PCF) is included in the TK5.1.1 delivery. It contains entries which are either required or optional for use of many Toolkit functions. This file is named $PGSRUN/PCF.relA.template. The particular example we use here is from the Pathfinder AVHRR/Land Toolkit Prototype study. The complete example file appears in Appendix B of this document.

It is recommended that you start with the same (customized) copy of the PCF each time you run at the SCF, especially if you are using temporary files in your processing. You don't want previous temporary file references in the PCF, since these files are deleted by the system (unless you are not using PGS_PC_Shell.sh or PGS_PC_TermCom).

The Unix environment variable $PGS_PC_INFO_FILE must point to your Process Control file in order for the Toolkit to work at all.

We go through the example file section-by-section. The sections of a Process Control file include:

- SYSTEM RUNTIME PARAMETERS
- PRODUCT INPUT
- PRODUCT OUTPUT
- SUPPORT INPUT
- SUPPORT OUTPUT
- USER-DEFINED RUNTIME PARAMETERS
- INTERMEDIATE INPUT
- INTERMEDIATE OUTPUT
- TEMPORARY IO

All sections of the PCF, except the SYSTEM RUNTIME PARAMETERS and USER- DEFINED RUNTIME PARAMETERS sections, consist of names, locations and other data about physical files. Each of these sections has a default file location, which is at the beginning of the section. The default file location is delimited by a '!' in column one of the PCF. This location points to the default directory in which these files are stored. This may be overridden for individual files, by inserting the fully-qualified physical directory path, as explained below. The PRODUCT INPUT section provides a detailed example of considerations that apply to all sections that involve files. Explanations of other sections provide only differences unique to those sections.

General considerations:

```
#   Process Control File: Pathfinder AVHRR/Land Toolkit
#      Prototype
#
#   Env variable PGS_PC_INFO_FILE must point to this file
```

Comments in a PCF are any lines that begin in the first column with "#".

```
?   SYSTEM RUNTIME PARAMETERS
```

The "**?**" symbol in the first column defines this line as the subject of the section. These nine subject names must not be changed nor deleted from the PCF.

Blank lines are not allowed.

Pipe character "|" must be used to delimit fields.

The exclamation point "!" must be used to designate the default file location. This must appear before any file entries in each section of the PCF.

The entire length of any line in the PCF may not exceed 1000 characters.

Different sections of the PCF have different numbers of required and optional fields for each entry. In the examples below, each entry is identified as required or optional.

3.1.2.1 SYSTEM RUNTIME PARAMETERS

```
?   SYSTEM RUNTIME PARAMETERS
# -----------------------------------------------------------
# Production Run ID - unique production run identifier
# -----------------------------------------------------------
1
```

This string identifies the particular run of your algorithm at the SDPS. **This field is required**, and may be up to 200 characters. It cannot be the string "0".

```
# -----------------------------------------------------------
# Software ID - unique software configuration
identifier
# -----------------------------------------------------------
1
```

This string identifies the particular software of which your PGE consists. **This field is required**, and may be up to 200 characters. It cannot be the string "0".

In the production system, both of these fields are written into the PCF by the SDP Planning and Scheduling sub-system. At the SCF, you may use any string you like. Note that the 'Production Run Id' value is used in the naming of Temporary and Intermediate files.

Currently these are the only two fields allowed in this section. DAAC and hardware identification are being considered as additions to the configuration data, for future deliveries of the Toolkit.

3.1.2.2 PRODUCT INPUT

This section is for primary data files used as input to create standard products. This includes such files as ancillary data, Level 0 data, and standard products output from other PGEs; in general, all of your input files.

```
?   PRODUCT INPUT FILES
# [ next line is for default location ]
! ~/runtime
```

Environment variable PGSHOME/runtime is the default location of the files in this section, unless it is overridden for individual files, as explained below. Note that the tilde character "~" is equated to the environment variable PGSHOME. This is true throughout the entire PCF. **This particular default file location $PGSHOME/runtime must not be changed**, because of the way the Toolkit Ancillary Data Access input files are handled. Default file locations of all other sections of the PCF may be changed to whatever you like.

```
# -----------------------------------------------------------
# Pathfinder AVHRR/Land input files
# -----------------------------------------------------------
201|87002002709.no9_gac|||||1
401|goldtopolandsea8.bin|||||1
402|gridtoms_1987_sngl_ntwk|||||1
403|ephem8788.dat|||||1
404|timecorr8788.dat|||||1
405|SDSannotations.dat|||||1
406|HDFmetadata.dat|||||1
410|jan021987.proclog|||||1
```

`201|87002002709.no9_gac|||||1`

The first entry in this section is used as an example; it is the primary input file for Pathfinder AVHRR/Land processing.

`201``|87002002709.no9_gac|||||1`

Field 1 is the link between your software and this PCF entry, the **logical identifier**. This identifier should be associated with a mnemonic in your code, at the beginning of the module where you use PGS_IO_Gen_Open to open this file, as shown below. **This field is required**, and must be an integer, of type PGSt_integer (long) in C, INTEGER in Fortran. Science software may use any positive integer for logical identifiers, except integers in the range 10,000-10,999; these numbers are reserved for the Toolkit.

In C, the form of this is

```
#define GAC_FILE 201
```

In Fortran,

```
      PARAMETER (GAC_FILE=201)
```

You then use GAC_FILE as an input parameter to Toolkit function PGS_IO_Gen_Open.

Note that while you can use hard-coded numbers in calling sequences, instead of mnemonics (C) or parameters (Fortran), this will make things difficult for integration and test, and also for maintainance; this practice is strongly discouraged.

```
201|87002002709.no9_gac||||1
```

Field 2 is the file **reference**, currently the actual physical filename, unqualified (i.e., without directory information). In the future production system, this mechanism may change (for example to a Universal Reference), but this will not affect the science software. **This field is required**, and is a string of up to 256 characters.

```
201|87002002709.no9_gac||||1
```

Field 3 is the **path name**, for overriding the default directory. In this example, the Toolkit will look for this file in location $PGSHOME/runtime /87002002709.no9_gac. If instead this entry were

```
201|87002002709.no9_gac|/fire2/toma/data||||1
```

then the Toolkit would look for this file in /fire2/toma/data/87002002709.no9_gac. This field is optional, and is a string of up to 100 characters.

```
201|87002002709.no9_gac||||1
```

Field 4, blank here, is reserved for future use.

```
201|87002002709.no9_gac||||1
```

Field 5, blank here, is the **universal reference**. It may contain any string of up to 150 characters. This value may be returned by calling the function PGS_PC_GetUniversalRef.

```
201|87002002709.no9_gac||||1
```

Field 6, blank here, is the **attribute location**. It is the name of a file that contains data about the file of Field 2. This file must be in the same directory as the file in Field 2. This field is optional, and is a string of up to 256 characters. For an example of an attribute file, see the descriptions of the PGS_PC_Get*Attr Tools.

```
201|87002002709.no9_gac||||1
```

Field 7 is the **sequence number**. It is used if there is more than one physical file associated with the logical identifier of Field 1, which is normally only the case for PRODUCT INPUT and PRODUCT OUTPUT files.
At the SCF, you must assign this sequence number to each instance of the file in the PCF; at the DAAC, this is done by the production system.
The actual value of the sequence number is not relevant to your code; it is an internal number used by the production system.
At the SCF, you **must** list these sequence numbers in the PCF starting with the largest first, then decrementing by one, down to the smallest (1), as shown in the example.

The **version number**, which is used as an argument to Toolkit functions that access different instances of a file, is not the same as sequence number.
The version number is the order which the files are listed in the PCF, from smallest (1) to largest.
As an example, if the PCF contains the entries

```
201|87002002710.no9_gac||||2
201|87002002709.no9_gac||||1
```

then file *87002002710.no9_gac* is version #1 (sequence #2), and file *87002002709.no9_gac* is version #2 (sequence #1).
No information about file content may be inferred from sequence number. This number is for internal system use only.
Use the version number, i.e., the order of listing of PCF entries, as the input to appropriate Toolkit functions.

(As you may have noticed, it so happens that the the version numbers specified in your code run opposite to the sequence numbers defined in the PCF.)

Field 7 is required for PRODUCT INPUT and PRODUCT OUTPUT files (but is optional for all other sections of the PCF). It must be an integer.

The rest of the entries in the PRODUCT INPUT section of the Pathfinder AVHRR/Land Toolkit Prototype file (Appendix B), in the section labeled "Toolkit product input files", are Toolkit files. These are normally not modified.

3.1.2.3 PRODUCT OUTPUT

This section is for standard product output files.

```
?   PRODUCT OUTPUT FILES
# [ next line is for default location ]
! ~/runtime
#
# ---------------------------------------------------------
# Pathfinder AVHRR/Land main output file
# ---------------------------------------------------------
301|test11.hdf|||||1
```

This file is defined in C code as

```
#define HDF_FILE      301
```

or in Fortran code as

```
      PARAMETER (HDF_FILE=301)
```

It resides in directory $PGSHOME/runtime. It does not have an attribute file.

This section has the same fields as PRODUCT INPUT.

3.1.2.4 SUPPORT INPUT

This section is primarily for files that are input to Toolkit functions. Ordinarily, you would not modify any entries in this section. An exception to this is the template files used for ancillary files; see the Ancillary Data Access Tools section.

```
?   SUPPORT INPUT FILES
# [ next line is for default location ]
! ~/runtime
#
# ---------------------------------------------------------
# Pathfinder AVHRR/Land support input files
# ---------------------------------------------------------
```

They reside in directory $PGSHOME/runtime. They have no attribute files.

This section has the same fields as PRODUCT INPUT and PRODUCT OUTPUT, except that Field 7 is not required>

There are no support input files in the Pathfinder AVHRR/Land Toolkit Prototype.

The entries in the SUPPORT INPUT section of the Pathfinder AVHRR/Land Toolkit Prototype file (Appendix B) are Toolkit files, mostly to support the Ancillary Data Access (AA) Tools. You may modify these, as explained in the AA Tools section of this document.

3.1.2.5 SUPPORT OUTPUT

This section is primarily for files that are output from Toolkit functions

```
?   SUPPORT OUTPUT FILES
# [ next line is for default location ]
! ~/runtime
#
```

This section has the same fields as PRODUCT INPUT and PRODUCT OUTPUT, except that Field 7 is not required.

There are no support output files in the Pathfinder AVHRR/Land Toolkit Prototype.

The Toolkit files in this section support the SMF Log files . You may change the names of the files and directories if you want, but not the logical identifier (Field 1).

3.1.2.6 USER-DEFINED RUNTIME PARAMETERS

This section of the PCF is different from the other sections in that it does not contain information about files. Instead, it may be used to obtain other kinds of information from the production environment.

```
?   USER DEFINED RUNTIME PARAMETERS
#
# ---------------------------------------------------------
# Pathfinder AVHRR/Land runtime parameters
# ---------------------------------------------------------
601|requested_size_x|409
602|requested_size_y|128
603|wait_time|3
```

601|requested_size_x|409

Field 1 is as always the **logical identifier**. **This field is required.**

```
601|requested_size_x|409
```

Field 2 is the **parameter name**. It is an optional text string of up to 200 characters. The Toolkit ignores this field; its intended use is for identification in this PCF, so you may enter whatever you like here. In the Pathfinder AVHRR/Land Toolkit Prototype, the name of the variable in the code is used for this purpose.

```
601|requested_size_x|409
```

Field 3 is the **parameter value**. This is read into your code as a string of up to 200 characters by the Toolkit (PGS_PC_GetConfigData). Your code is responsible for any necessary conversion. e.g. to integer. **This field is required**.

Toolkit files in this section support the sending of files and email to remote locations. For an explanation of these entries in the PCF, see the Notes section of the Tool Description for PGS_SMF_SendRuntimeData.

3.1.2.7 INTERMEDIATE INPUT

```
?   INTERMEDIATE INPUT
# [ next line is for default location]
! ~/runtime
#
```

This section and the next are for intermediate files, or files that will exist for longer than a single PGE, but are not standard products. This section is for intermediate input files.

This section has the same fields as PRODUCT INPUT and PRODUCT OUTPUT, except that Field 7 is not required. The unqualified file name (Field 2) is ordinarily the name of a file that was generated as an INTERMEDIATE OUTPUT file by a previous run of this PGE. At the SCF, if you are testing successive runs of a PGE which share intermediate files, you need to make sure that the logical identifier is the same in the PCF that you use for all the runs. If you are accessing the intermediate file from a different PGE than the one that created it, you also need to make sure that the mnemonic definitions in your code reference the same logical identifier. You should also copy over the file name if you use a different PCF.

How intermediate files are handled in the production environment, specifically how long they stay around, has not been determined at this writing.

Use function PGS_IO_Gen_Temp_Open (C) or PGS_IO_Gen_Temp_OpenF (Fortran) to open intermediate files.

There are no intermediate files in the Pathfinder AVHRR/Land Toolkit Prototype.

3.1.2.8 INTERMEDIATE OUTPUT

```
?   INTERMEDIATE OUTPUT
# [next line is for default location]
! ~/runtime
#
```

This section is for intermediate output files.

Entries for this section of the PCF are created by the Toolkit; you do not need to enter values.

This section has the same fields as PRODUCT INPUT and PRODUCT OUTPUT, except that Field 7 is not required. The unqualified file name (Field 2) is generated by the Toolkit.

How intermediate files are handled in the production environment, specifically how long they stay around, has not been determined at this writing.

Use function PGS_IO_Gen_Temp_Open (C) or PGS_IO_Gen_Temp_OpenF (Fortran) to open intermediate files.

There are no intermediate files in the Pathfinder AVHRR/Land Toolkit Prototype.

3.1.2.9 TEMPORARY IO

Temporary files are files that exist only for the duration of a single PGE; the production system deletes these files automatically on PGE termination. (You may use the function PGS_IO_Gen_Temp_Delete to do this at the SCF.) Since a single PGE may consist of several of your executables, this section is part of the PCF to enable these files to be passed among these executables.

Entries for this section of the PCF are created by the Toolkit; you do not need to enter values. The unqualified file name (Field 2) is generated by the Toolkit.

```
?   TEMPORARY IO
# [ next line is for default location ]
! ~/runtime
#
# ------------------------------------------------------------
# Pathfinder AVHRR/Land temporary file
# ------------------------------------------------------------
901|pc1157318822894183312||0|0|0|0
```

This file is defined in C code as

```
#define BINARY_OUTPUT               901
```

or in Fortran code as

```
      PARAMETER (BINARY_OUTPUT=901)
```

It resides in directory $PGSHOME/runtime. It has no attribute file.

If you are sharing this temporary file among executables in the same PGE, then you need to have the same #define or PARAMETER statement in the code for each appropriate executable.

Use function PGS_IO_Gen_Temp_Open (C) or PGS_IO_Gen_Temp_OpenF (Fortran) to open temporary files. Use function PGS_IO_Gen_Temp_Delete to delete files you no longer need within a PGE.

This section has the same fields as PRODUCT INPUT and PRODUCT OUTPUT, except that Field 7 is not required.

3.1.2.10 End of PCF

All PCFs must end with the line

```
? END
```

Any information after this line is ignored.

3.1.3 Checking your Process Control File

Now that you have created your PCF, you can use it from your software through use of the Toolkit. However, you might want to check it to see if you have entered everything correctly. You can do this by using the pccheck utility, a Unix executable included with the Toolkit. This program is compiled at the time of Toolkit installation, and is located in directory $PGSBIN. You execute shell script **pccheck.sh**, which calls executable **pctcheck**; its source code is $PGSSRC/PC/PGS_PC_Check.c . To run it on your file *mypcfile*, on the Unix command line type

```
$PGSBIN/pccheck.sh -i mypcfile
```

If there are any errors in your file, you will see messages of the form

```
Error - problem with version number in Standard input file
Line number:  23
Line:  401|goldtopolandsea8.bin|||||
```

In this example the version number was omitted from the STANDARD INPUT file entry.

At the end, you will see a summary of the form

```
Check of mypcfile completed
Errors found:  7
Warnings found:  0
```

For this utility, a **pccheck error** is defined as a PCF entry that will cause a Toolkit PC function to return an error message. A **pccheck warning** is defined as an incorrect entry that will not cause the Toolkit trouble, but may cause the PGE to operate incorrectly. For example, a blank character in the file name field does not bother a Toolkit PC function, since it simply returns the string as is; **pccheck** will not return an error. But a blank character will certainly cause a Unix error, when the file open is attempted by a Toolkit function; **pccheck** will return a warning to this effect. Output is returned to stdout (usually the screen).

This is a simple explanation of how the pccheck utility works. For details, including a list of error messages, and information about other command line options, see "Validating Process Control Files", sec. C.2 of Appendix C, in the Toolkit Users Guide.

3.1.4 Metadata considerations

Protoype metadata (MET) tools, which format standard product metadata for ingest **to** the data server(SDS), will be available in the next Toolkit delivery.

For the present, the only Toolkit functions that deal with data about data are the tools PGS_PC_GetFileAttr, PGS_PC_GetFileAttrCom and PGS_PC_GetFileByAttr. These functions are involved in retrieving file "attribute" data **from** the system, via the Process Control file. Essentially, you can get character string metadata from a text file using these functions.

Since details about how the production system handles metadata are not yet available, this mechanism was determined to be the best that can be done about this issue at the moment. Every effort will be made to keep the calling sequence unchanged for these tools in the future. However, given the uncertainties about this issue, this cannot be guaranteed.

3.1.5 Command tools for use in shell scripts

This section briefly describes the usage of the set of Command tools, which are callable from Unix shell scripts. These tools are generally identified by the suffix "Com" in the function name.

Tool PGS_PC_Shell.sh is used to call your PGE. It is strongly recommended that you call your PGE from this function during testing at the SCF; among other things, it enables Toolkit (not user) shared memory, which speeds execution of certain Toolkit functions.
You must use either this tool or *PGS_PC_InitCom* if you want to enable the creation of Toolkit log files.
You must use either this tool or *PGS_PC_TermCom* if you want to send any files to a remote machine, through use of function PGS_SMF_SendRuntimeData.

Tools PGS_PC_InitCom and PGS_PC_TermCom are used to initialize and terminate your PGE respectively.
Ordinarily you would not use these, as they are called internally by *PGS_PC_Shell.sh*. They are included in the Toolkit documentation for reference in case you wish to customize *PGS_PC_Shell.sh* for some reason; however, please note that any such customization is not part of your delivery to the DAAC.

Note: The above three functions are used at the DAAC as well as at your SCF; however, it is not necessary to include them in scripts that you deliver to the DAAC for Integration and Test. They are included as part of the Toolkit delivery for your use in testing at the SCF.
The three functions are to be used *outside* of your PGE.

The rest of the PGS_PC_*Com tools are simply wrappers on Toolkit API tools. These functions are for use *inside* your PGE.

Further details are given in the Tool Descriptions.

3.2 Process Control (PC) Tool descriptions

3.2.1 PGS_PC_InitCom

Short explanation of what it's for: Command line function for initializing the Toolkit for use with your PGE.
Normally not used at the SCF, since its functionality is fully covered by PGS_PC_Shell.sh.

This function is in file: $PGSSRC/PC/PGS_PC_InitCom.c

Shell example:

```
# Execute a PGE, enabling Toolkit (not user) shared memory,
#    and also initializing creation of Toolkit log files,
# and have an SMF Cache size of 50.

unix% PGS_PC_InitCom 1 1 50
```

Notes:

You might want to use this function if you decide to write a custom script to call your PGE, in lieu of using PGS_PC_Shell.sh; however, please note that any such customization is not part of your delivery to the DAAC.

This function enables your PGE to

- Use Toolkit (not user) shared memory, which speeds up Toolkit processing
- Automatically load the Process Control File into Toolkit shared memory
- Automatically create Toolkit log files

The first argument of this function is for turning on or off Toolkit (not user) shared memory, if available; the second argument is for turning on or off creation of Toolkit log files. The third argument is to specify the amount (in records) of SMF Cache memory to reserve for the storage of SMF messages.

In the example "unix%" is the Unix command line prompt.

For this particular function, the example is identical for any Unix shell.

3.2.2 PGS_PC_GenUniqueID

Short explanation of what it's for: Generates a string that uniquely identifies your standard product output file. May be used as file metadata.

This function is in file: $PGSSRC/PC/PGS_PC_GenUniqueID.c

Examples:

The examples assume the following exist in the Process Control File (PCF):

```
?   SYSTEM RUNTIME PARAMETERS
# -----------------------------------------------------------
# Production Run ID - unique production instance identifier
# -----------------------------------------------------------
1
# -----------------------------------------------------------
# Software ID - unique software configuration identifier
# -----------------------------------------------------------
1
```

C example:

```
#include <PGS_PC.h>
#define HDF_FILE      301
char uniqueID[PGSd_PC_LABEL_SIZE_MAX];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
returnStatus = PGS_PC_GenUniqueID(HDF_FILE,uniqueID);
/*
Variable uniqueID now contains the string
"PRID - 1 SID - 1 PRODID - 301"
*/
```

Fortran example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INTEGER pgs_pc_genuniqueid
      INTEGER HDF_FILE
      PARAMETER(HDF_FILE=301)
      CHARACTER*200 uniqueid
      INTEGER returnstatus
C
C Begin example
C
      returnstatus = pgs_pc_genuniqueid(HDF_FILE,uniqueid)
C
C Variable uniqueid now contains the string
C 'PRID - 1 SID - 1 PRODID - 301'
C
```

Notes:

The mechanism for using the output of this function as file metadata has not yet been defined.

4. Generic I/O (IO_Gen) Tools

4.1 Overview

This section describes the Generic I/O (IO_Gen) Tools. These tools are used in your code where appropriate to open, close and delete various files, such as temporary, intermediate, and other miscellaneous files. They are also used by the Toolkit to access ancillary, Level 0 and other files.

These tools are unique in the Toolkit in that there are different versions of the source code, and different calling sequences for C and FORTRAN for each tool (except PGS_IO_Gen_Temp_Delete); in contrast to the rest of the Toolkit, which is written in C with FORTRAN bindings. This is necessary because of the different ways C and FORTRAN define file handles, since these file handles are input to the native C and FORTRAN I/O functions such as fscanf and READ. FORTRAN functions are identified by the suffix "F".

A couple of definitions are in order: a **temporary file** is one that exists only for the duration of a single PGE, but may be shared between executable modules within that PGE; an **intermediate file** is one that may stick around for a user-defined time.

Special note regarding HDF: No Toolkit functions exist yet to access HDF files; currently you are to use the native NCSA functions for HDF access. To see how to get the physical filename needed as input to the NCSA HDF open file function, see the example for Toolkit function PGS_PC_GetReference .

4.2 Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_IO_Gen_* tools.

4.2.1 PGS_IO_Gen_Close

Short explanation of what it's for: Close a file that was opened by PGS_IO_Gen_Open or PGS_IO_Gen_Temp_Open (C version).

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_Close.c

Examples:

C example:

```
#include <PGS_IO.h>
PGSt_IO_Gen_FileHandle *handle;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
returnStatus = PGS_IO_Gen_Close( handle );
```

FORTRAN example:

This function is not callable from FORTRAN. See PGS_IO_Gen_CloseF .

Notes:

The Toolkit internally keeps track of which files have been opened and closed.

4.2.2 PGS_IO_Gen_CloseF

Short explanation of what it's for: Close a file that was opened by PGS_IO_Gen_OpenF or PGS_IO_Gen_Temp_OpenF (FORTRAN version).

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_CloseF.f

Examples:

C example:

This function is not callable from C. See PGS_IO_Gen_Close.

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER pgs_io_gen_closef
      INTEGER handle
      INTEGER returnstatus
C
C Begin example
C
      returnstatus = pgs_io_gen_closef(handle)
```

Notes:

The Toolkit internally keeps track of which files have been opened and closed.

4.2.3 PGS_IO_Gen_Open

Short explanation of what it's for: Open a generic file (C version). Intended for use in opening miscellaneous files in your software (see Notes).

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_Open.c

Examples:

The example assumes the following exists in the Process Control File (PCF):

```
?   PRODUCT INPUT FILES
# -----------------------------------------------------------
# Pathfinder AVHRR/Land input files
# -----------------------------------------------------------
401|goldtopolandsea21.bin||||2
401|goldtopolandsea33.bin||||1
```

C example:

```
#include <stdio.h>
#include <PGS_IO.h>
#define GOLDEN_BINARY 401
PGSt_IO_Gen_FileHandle *processGolden;
PGSt_integer version;
long xScale;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
   Open a file for read
*/
version = 1;
returnStatus = PGS_IO_Gen_Open( GOLDEN_BINARY,
                  PGSd_IO_Gen_Read, &processGolden, version);
/*
   The file $PGS_PRODUCT_INPUT/goldtopolandsea21.bin is now open
      (see Notes)
*/
/*
    File handle variable processGolden may now be used as
    an argument to any native C I/O function that takes a
    variable of type FILE as input, e.g.,
*/
fscanf( processGolden, "%ld", &xScale );
```

FORTRAN example:

This function is not callable from FORTRAN. See PGS_IO_Gen_OpenF .

Notes:

Use NCSA HDF open function *Hopen* to open standard product files. Use other Toolkit functions to open Ancillary and Level 0 files. Use PGS_IO_Gen_Temp_Open to open temporary and intermediate files. This function is for opening any other files.

In the example, the user requested **version 1** of the GOLDEN_BINARY file to be opened. This refers to the **first** entry in the PC file, i.e., file *goldtopolandsea21.bin*. The sequence numbers in the PC file are in reverse order from the version numbers used in arguments to Toolkit functions. So in the PC file, the entry for file *goldtopolandsea21.bin* has sequence number 2. For further explanation of sequence numbers in the PC file, see section 4.1.2.2 of the Process Control Overview, "PRODUCT INPUT".

The Toolkit internally keeps track of which files have been opened and closed.

A valid Process Control file (PCF) must have been constructed before using this tool. See section 4, "Process Control (PC) Tools".

The following is a complete listing of the access modes available (2nd argument in calling sequence):

PGS_IO_Gen_Open Access Modes

```
    Toolkit                 C        Description


PGSd_IO_Gen_Read         "r"   Open file for reading
PGSd_IO_Gen_Write        "w"   Open file for writing, truncating
                                existing file to 0 length, or
                                creating a new file
PGSd_IO_Gen_Append       "a"   Open file for writing, appending
                                to the end of existing file, or
                                creating file
PGSd_IO_Gen_Update       "r+"  Open file for reading and writing
PGSd_IO_Gen_Trunc        "w+"  Open file for reading and writing,
                                truncating existing file to zero
                                length, or creating new file
PGSd_IO_Gen_AppendUpdate "a+"  Open file for reading and writing,
                                appending to the end of existing
                                file, or creating a new file;
                                whole file can be read, but
                                writing only appended
```

ToolkitMnemonic used as 2nd argument in calling sequenceCEquivalent access mode for native POSIX C function fopenDescriptionToolkit access mode description

!!!!!!!!!!! During testing of this tool, the mode AppendUpdate (a+)
!! **ALERT** !! was found to produce results that were not consistent
!!!!!!!!!!! with the documented POSIX standard.The sort of behavior
that was typically observed was for data, buffered during a read
operation, to be appended to the file along with other data that was
being written to the file. Note that this behavior could not be attributed
to the Toolkit since the same behavior was revealed when purely "POSIX"
calls were used.

If you are using the Toolkit LogStatus log file, you may see a sequence of messages like this:

```
PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group

PGS_PC_GetPCSDataLocateEntry():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group

PGS_PC_GetPCSData():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group
```

The presence of these messages is an artifact of the way the Toolkit access the Process Control file. Up to 4 sets of these messages (12 total) are generated each time PGS_IO_Gen_Open is called.
A means of limiting this to one set of messages will be in place by the TK5 delivery of July 1995.

4.2.4 PGS_IO_Gen_OpenF

Short explanation of what it's for: Open a generic file (FORTRAN version). Intended for use in opening miscellaneous files in your software (see Notes).

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_OpenF.f (f77 version), $PGSSRC/IO/GEN/PGS_IO_Gen_OpenF90.f (F90 version).

Examples:

The example assumes the following exists in the Process Control File (PCF):

```
?   PRODUCT INPUT FILES
# ----------------------------------------------------------
# Pathfinder AVHRR/Land input files
# ----------------------------------------------------------
401|goldtopolandsea21.bin||||2
401|goldtopolandsea33.bin||||1
```

C example:

This function is not callable from C. See PGS_IO_Gen_Open .

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER GOLDEN_BINARY
      PARAMETER (GOLDEN_BINARY=401)
      INTEGER pgs_io_gen_openf
      INTEGER processgolden
      INTEGER version
      INTEGER xscale
      INTEGER returnstatus
C

C Begin example
C
C Open a sequential unformatted file for read
C
      version = 1
      returnstatus = pgs_io_gen_openf( GOLDEN_BINARY,
     .         PGSd_IO_Gen_RSeqUnf, 0, processgolden, version)
C
C The file $PGS_PRODUCT_INPUT/goldtopolandsea21.bin is now open
C     (see Notes)
C
C File handle variable processGolden may now be used as
C    an argument to any native I/O FORTRAN function, e.g.,
C
      READ(processGolden) xscale
```

Notes:

Use NCSA HDF open function *Hopen* to open standard product files. Use other Toolkit functions to open Ancillary and Level 0 files. Use PGS_IO_Gen_Temp_OpenF to open temporary and intermediate files. This function is for opening any other files.

All FORTRAN access modes are supported. For an example of using direct access files, see the examples for function PGS_IO_Gen_Temp_Open.

The 3rd argument of PGS_IO_Gen_OpenF is for specifying record length.
In FORTRAN 77, this value must be at least 1 for direct access files; it is ignored for sequential files (as in the example).
In FORTRAN 90, this value must be at least 1 for direct access files. For sequential access, if this value is 0, the file is opened with a platform-dependent record length; otherwise, it is opened with the specified record length.

Files which are opened with one of the **direct access** modes must have been created by FORTRAN direct access writes. That is, you cannot expect to read a file with direct access reads if the file is sequential.

In the example, the user requested **version 1** of the GOLDEN_BINARY file to be opened. This refers to the **first** entry in the PC file, i.e., file *goldtopolandsea21.bin*. The sequence numbers in the PC file are in reverse order from the version numbers used in arguments to Toolkit functions. So in the PC file, the entry for file *goldtopolandsea21.bin* has sequence number 2. For further explanation of sequence numbers in the PC file, see section 4.1.2.2 of the Process Control Overview, "PRODUCT INPUT".

The Toolkit internally keeps track of which files have been opened and closed.

A valid Process Control file (PCF) must have been constructed before using this tool. See section 4, "Process Control (PC) Tools".

The following is a complete listing of the access modes available (3rd argument in calling sequence):

PGS_IO_Gen_OpenF Access Modes

```
                                     ****FORTRAN****
    Toolkit             mode      'access='     'form='
PGSd_IO_Gen_RSeqFrm   Read    Sequential   Formatted
PGSd_IO_Gen_RSeqUnf   Read    Sequential   Unformatted
PGSd_IO_Gen_RDirFrm   Read    Direct       Formatted
PGSd_IO_Gen_RDirUnf   Read    Direct       Unformatted

PGSd_IO_Gen_WSeqFrm   Write   Sequential   Formatted
PGSd_IO_Gen_WSeqUnf   Write   Sequential   Unformatted
PGSd_IO_Gen_WDirFrm   Write   Direct       Formatted
PGSd_IO_Gen_WDirUnf   Write   Direct       Unformatted

PGSd_IO_Gen_USeqFrm   Update  Sequential   Formatted
PGSd_IO_Gen_USeqUnf   Update  Sequential   Unformatted
PGSd_IO_Gen_UDirFrm   Update  Direct       Formatted
PGSd_IO_Gen_UDirUnf   Update  Direct       Unformatted

The following modes are available in FORTRAN 90 only:

PGSd_IO_Gen_ASeqFrm   Append  Sequential   Formatted
PGSd_IO_Gen_ASeqUnf   Append  Sequential   Unformatted
```

ToolkitMnemonic used as 2nd argument in calling sequencemodeType of access allowed'access='Equivalent argument of ACCESS parameter in FORTRAN OPEN function'form='Equivalent argument of FORM parameter in FORTRAN OPEN function

If you are using the Toolkit LogStatus log file, you may see a sequence of messages like this:

```
PGS_PC_GetPCSDataGetIndex():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group

PGS_PC_GetPCSDataLocateEntry():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group

PGS_PC_GetPCSData():PGSPC_W_NO_FILES_EXIST:76802
No files exist for product group
```

The presence of these messages is an artifact of the way the Toolkit access the Process Control file. Up to 4 sets of these messages (12 total) are generated each time PGS_IO_Gen_OpenF is called.
A means of limiting this to one set of messages will be in place by the TK5 delivery of July 1995.

This function corresponds to two similar but separate source code files, one for FORTRAN 77, and one for FORTRAN 90. The FORTRAN 90 version has all the functionality of the FORTRAN 77 version, plus support for (1) Append mode and (2) specification of record length for sequential files.
The installation script compiles one of these versions, based on which flavor of FORTRAN you specified at the time of Toolkit installation.

4.2.5 PGS_IO_Gen_Temp_Delete

Short explanation of what it's for: Mark a temporary file for deletion. You would use this tool if you want to re-use a logical ID used by a temporary file you no longer need, to open a new temporary file.

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_Temp_Delete.c.

Examples:

C example:

```
#include <PGS_IO.h>
#define TEMP_BINARY 901
PGSt_SMF_status returnStatus;

returnStatus = PGS_IO_Gen_Temp_Delete( TEMP_BINARY );

/* The file corresponding to logical ID TEMP_BINARY is
   now marked for deletion; the logical ID may be used to
   open another file using PGS_IO_Gen_Temp_Open. */
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER TEMP_BINARY
      PARAMETER (TEMP_BINARY=901)
      INTEGER pgs_io_gen_temp_delete
      INTEGER returnstatus
C
C Begin example
C
      returnstatus = pgs_io_gen_temp_delete( TEMP_BINARY )
C The file corresponding to logical ID TEMP_BINARY is
C  now marked for deletion; the logical ID may be used to
C  open another file using PGS_IO_Gen_Temp_OpenF.
```

Notes:

This function is only for use with Temporary files, and not Intermediate files.

This function merely marks Temporary files for deletion, so you can re-use the same logical ID. It does not physically delete files.

If you are using PGS_PC_Shell.sh to wrap your PGE, then your Temporary files are automatically deleted at PGE termination.
If you are not wrapping your PGE with the shell, you should delete your Temporary files manually before each of your test runs. In addition, you need to start with a fresh Process Control file, i.e., the PCF at the beginning of any run should have no entries in the TEMPORARY I/O section.
In the production environment, Temporary files are always deleted automatically at the end of your PGE (since PGS_PC_Shell.sh is used there).

This is the only PGS_IO_Gen function that has no separate FORTRAN version. FORTRAN access is provided through bindings to the C code, as in the rest of the Toolkit.

4.2.6 PGS_IO_Gen_Temp_Open

Short explanation of what it's for: Open a temporary or intermediate file (C version). Temporary files exist for the duration of one PGE only; intermediate files may have a longer duration.

This function is in file: $PGSSRC/IO/GEN/PGS_IO_Gen_Temp_Open.c.

Examples:

The example assumes the following exists in the Process Control File (PCF):

```
?   INTERMEDIATE INPUT
# [set env var PGS_INTERMEDIATE_INPUT for default location]
701|pc11502832010280003951O4034|||||
#
?   INTERMEDIATE OUTPUT
# [set env var PGS_INTERMEDIATE_OUTPUT for default location]
#
?   TEMPORARY IO
# [set env var PGS_TEMPORARY_IO for default location]
#
?   END
```

C example:

```
#include <PGS_IO.h>
#define INTERMEDIATE_IN 701
#define INTERMEDIATE_OUT 801
#define TEMP_BINARY 901
PGSt_IO_Gen_FileHandle *handle;
long xScale;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
Open the existing intermediate input file for read
Read a value
Close it
*/
returnStatus = PGS_IO_Gen_Temp_Open( PGSd_IO_Gen_Endurance,
                  INTERMEDIATE_IN, PGSd_IO_Gen_Read, &handle );
fscanf( handle, "%ld", &xScale );
returnStatus = PGS_IO_Gen_Close( handle );
/*
You have just read a value from file
$PGS_INTERMEDIATE_INPUT/pc1150283201028000395104034
*/
/*
Open a new intermediate output file for write
Write a value
Close it
*/
returnStatus = PGS_IO_Gen_Temp_Open( PGSd_IO_Gen_Endurance,
                 INTERMEDIATE_OUT, PGSd_IO_Gen_Write, &handle );
fprintf( handle, "%ld", xScale );
returnStatus = PGS_IO_Gen_Close( handle );
/*
You have just written a value to a new file in directory
$PGS_INTERMEDIATE_OUTPUT
*/
/*
Open a temporary file for write
Write a value
Close it
*/
returnStatus = PGS_IO_Gen_Temp_Open( PGSd_IO_Gen_NoEndurance,
                 TEMP_BINARY, PGSd_IO_Gen_Write, &handle );
fprintf( handle, "%ld", xScale );
returnStatus = PGS_IO_Gen_Close( handle );
/*
You have just written a value to a new file in directory
$PGS_TEMPORARY_IO
*/
```

FORTRAN example:

This function is not callable from FORTRAN. See PGS_IO_Gen_Temp_OpenF .

Notes:

The difference between this function and PGS_IO_Gen_Open is that this tool enables tracking of temporary and intermediate files in the production system, as well as providing for file name generation.

The following applies to Temporary, not Intermediate, files:

Process Control file entries for Temporary files are generated automatically by the Toolkit. You should never create a PCF entry for a Temporary file.

If you are using PGS_PC_Shell.sh to wrap your PGE, then your Temporary files are automatically deleted at PGE termination.
If you are not wrapping your PGE with the shell, you should delete your Temporary files manually before each of your test runs. In addition, you need to start with a fresh Process Control file, i.e., the PCF at the beginning of any run should have no entries in the TEMPORARY I/O section.
In the production environment, Temporary files are always deleted automatically at the end of your PGE (since PGS_PC_Shell.sh is used there).

The first argument of PGS_IO_Gen_Temp_Open specifies whether the file to open is Temporary or Intermediate. Currently this is a simple Boolean value. In the future this argument may be changed to specify the duration of an intermediate file. The calling sequence will not change.

After the Toolkit calls given in the example have been executed, the last 3 sections of the Process Control file will look like this:

```
?   INTERMEDIATE INPUT
# [set env var PGS_INTERMEDIATE_INPUT for default location]
701|pc1150283201028000395104034|||||
#
?   INTERMEDIATE OUTPUT
# [set env var PGS_INTERMEDIATE_OUTPUT for default location]
801|pc1150283201039509195162200|||||
#
?   TEMPORARY IO
# [set env var PGS_TEMPORARY_IO for default location]
901|pc1150283201039509195162229|||||
#
?   END
```

-- the toolkit has created file names for the new files and stored them in the PCF.

See the Notes section of tool PGS_PC_GetTempReferenceCom for an explanation of the form of the temporary file names.

A valid Process Control file (PCF) must have been constructed before using this tool. See section 4, "Process Control (PC) Tools".

The following is a complete listing of the access modes available (3rd argument in calling sequence):

PGS_IO_Gen_Temp_Open Access Modes

```
    Toolkit                 C         Description


PGSd_IO_Gen_Read         "r"   Open file for reading
PGSd_IO_Gen_Write        "w"   Open file for writing, truncating
                                existing file to 0 length, or
                                creating a new file
PGSd_IO_Gen_Append       "a"   Open file for writing, appending
                                to the end of existing file, or
                                creating file
PGSd_IO_Gen_Update       "r+"  Open file for reading and writing
PGSd_IO_Gen_AppendUpdate "a+"  Open file for reading and writing,
                                appending to the end of existing
                                file, or creating a new file;
                                whole file can be read, but
                                writing only appended
```

ToolkitMnemonic used as 3rd argument in calling sequenceCEquivalent access mode for native POSIX C function fopenDescriptionToolkit access mode description

!!!!!!!!!!! During testing of this tool, the mode AppendUpdate (a+)
!! **ALERT** !! was found to produce results that were not consistent
!!!!!!!!!!! with the documented POSIX standard.The sort of behavior
that was typically observed was for data, buffered during a read
operation, to be appended to the file along with other data that was
being written to the file. Note that this behavior could not be attributed
to the Toolkit since the same behavior was revealed when purely "POSIX"
calls were used.

4.2.7 PGS_IO_Gen_Temp_OpenF

Short explanation of what it's for: Open a temporary or intermediate file (FORTRAN version). Temporary files exist for the duration of one PGE only; intermediate files may exist for a longer duration.

This function is in file:
$PGSSRC/IO/GEN/PGS_IO_Gen_Temp_OpenF.f (F77 version),
$PGSSRC/IO/GEN/PGS_IO_Gen_Temp_OpenF90.f (F90 version).

Examples:

The example assumes the following exists in the Process Control File (PCF):

```
?   INTERMEDIATE INPUT
# [set env var PGS_INTERMEDIATE_INPUT for default location]
701|pc1150283201028000395104034|||||
#
?   INTERMEDIATE OUTPUT
# [set env var PGS_INTERMEDIATE_OUTPUT for default location]
#
?   TEMPORARY IO
# [set env var PGS_TEMPORARY_IO for default location]
#
?   END
```

C example:

This function is not callable from C. See PGS_IO_Gen_Temp_Open .

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'
      INTEGER INTERMEDIATE_IN
      PARAMETER (INTERMEDIATE_IN=701)
      INTEGER INTERMEDIATE_OUT
      PARAMETER (INTERMEDIATE_OUT=801)
      INTEGER TEMP_BINARY
      PARAMETER (TEMP_BINARY=901)
      INTEGER pgs_io_gen_temp_openf
      INTEGER pgs_io_gen_closef
      INTEGER handle
      INTEGER xscale
      INTEGER recordlength
      INTEGER returnstatus
C
C Begin example
C
C Open the existing intermediate sequential unformatted
C     input file for read
C Read a value
C Close it
C
      returnstatus = pgs_io_gen_temp_openf( PGSd_IO_Gen_Endurance,
     .   INTERMEDIATE_IN, PGSd_IO_Gen_RSeqUnf, 0, handle )
      READ(handle) xscale
      returnstatus = pgs_io_gen_closef( handle )
C
C You have just read a value from file
C $PGS_INTERMEDIATE_INPUT/pc11502832010280000395104034
C
C Open a new intermediate direct access unformatted
C     output file for write
C Write a value
C Close it
C
      recordlength = 4
      returnstatus = pgs_io_gen_temp_openf(
     .    PGSd_IO_Gen_Endurance, INTERMEDIATE_OUT,
     .    PGSd_IO_Gen_WDirUnf, recordlength, handle )
      WRITE( handle, REC=1 ) xscale
      returnstatus = pgs_io_gen_close( handle )
C
C You have just written a value to a new file in directory
C $PGS_INTERMEDIATE_OUTPUT
C
C Open a temporary sequential formatted file for write
C Write a value
C Close it
C
      returnstatus = pgs_io_gen_temp_openf(
     .    PGSd_IO_Gen_NoEndurance, TEMP_BINARY,
     .    PGSd_IO_Gen_WSeqFrm , 0, handle )
      WRITE( handle, 100 ) xscale
100   FORMAT(I6)
      returnstatus = pgs_io_gen_close( handle )
C
C You have just written a value to a new file in directory
C $PGS_TEMPORARY_IO
C
```

Notes:

The difference between this function and PGS_IO_Gen_Open is that this tool enables tracking of temporary and intermediate files in the production system, as well as providing for file name generation.

The following applies to Temporary, not Intermediate, files:

Process Control file entries for Temporary files are generated automatically by the Toolkit. You should never create a PCF entry for a Temporary file.

If you are using PGS_PC_Shell.sh to wrap your PGE, then your Temporary files are automatically deleted at PGE termination.
If you are not wrapping your PGE with the shell, you should delete your Temporary files manually before each of your test runs. In addition, you need to start with a fresh Process Control file, i.e., the PCF at the beginning of any run should have no entries in the TEMPORARY I/O section.
In the production environment, Temporary files are always deleted automatically at the end of your PGE (since PGS_PC_Shell.sh is used there).

The first argument of PGS_IO_Gen_Temp_Open specifies whether the file to open is Temporary or Intermediate. Currently this is a simple Boolean value. In the future this argument may be changed to specify the duration of an intermediate file. The calling sequence will not change.

After the Toolkit calls given in the example have been executed, the last 3 sections of the Process Control file will look like this:

```
?   INTERMEDIATE INPUT
# [set env var PGS_INTERMEDIATE_INPUT for default location]
701|pc11502832010280003951040 34|||||
#
?   INTERMEDIATE OUTPUT
# [set env var PGS_INTERMEDIATE_OUTPUT for default location]
801|pc115028320103950919516 2200|||||
#
?   TEMPORARY IO
# [set env var PGS_TEMPORARY_IO for default location]
901|pc115028320103950919516 2229|||||
#
?   END
```

-- the toolkit has created file names for the new files and stored them in the PCF.

See the Notes section of tool PGS_PC_GetTempReferenceCom for an explanation of the form of the temporary file names.

All FORTRAN access modes are supported.

The 4th argument of PGS_IO_Gen_Temp_OpenF is for specifying record length.

In FORTRAN 77, this value must be at least 1 for direct access files; it is ignored for sequential files (as in the example).
In FORTRAN 90, this value must be at least 1 for direct access files. For sequential access, if this value is 0, the file is opened with a platform-dependent record length; otherwise, it is opened with the specified record length.

Files which are opened with one of the **direct access** modes must have been created by FORTRAN direct access writes. That is, you cannot expect to read a file with direct access reads if the file is sequential.

Temporary files are deleted automatically at PGE termination in the production environment. You may also delete them sooner by using tool PGS_IO_Gen_Temp_Delete.

A valid Process Control file (PCF) must have been constructed before using this tool. See section 4, "Process Control (PC) Tools".

The following is a complete listing of the access modes available (3rd argument in calling sequence):

PGS_IO_Gen_Temp_OpenF Access Modes

```
                                      ****FORTRAN****
    Toolkit             mode       'access='     'form='
PGSd_IO_Gen_RSeqFrm   Read     Sequential   Formatted
PGSd_IO_Gen_RSeqUnf   Read     Sequential   Unformatted
PGSd_IO_Gen_RDirFrm   Read     Direct       Formatted
PGSd_IO_Gen_RDirUnf   Read     Direct       Unformatted

PGSd_IO_Gen_WSeqFrm   Write    Sequential   Formatted
PGSd_IO_Gen_WSeqUnf   Write    Sequential   Unformatted
PGSd_IO_Gen_WDirFrm   Write    Direct       Formatted
PGSd_IO_Gen_WDirUnf   Write    Direct       Unformatted

PGSd_IO_Gen_USeqFrm   Update   Sequential   Formatted
PGSd_IO_Gen_USeqUnf   Update   Sequential   Unformatted
PGSd_IO_Gen_UDirFrm   Update   Direct       Formatted
PGSd_IO_Gen_UDirUnf   Update   Direct       Unformatted

The following modes are available in FORTRAN 90 only:

PGSd_IO_Gen_ASeqFrm   Append   Sequential   Formatted
PGSd_IO_Gen_ASeqUnf   Append   Sequential   Unformatted
```

ToolkitMnemonic used as 2nd argument in calling sequencemodeType of access allowed'access='Equivalent argument of ACCESS parameter in FORTRAN OPEN function'form='Equivalent argument of FORM parameter in FORTRAN OPEN function

This function corresponds to two similar but separate source code files, one for FORTRAN 77, and one for FORTRAN 90. The FORTRAN 90 version has all the functionality of the FORTRAN 77 version, plus support for (1) Append mode and (2) specification of record length for sequential files.
The installation script compiles one of these versions, based on which flavor of FORTRAN you specified at the time of Toolkit installation.

5. Memory Management (MEM) Tools

5.1 Memory Management (MEM) Tools Overview

The Memory Management group of tools allocates memory in your code.

There are two distinct sets of these tools: (1) the **dynamic memory** allocation tools, and (2) the **shared memory** allocation tools.

5.1.1 Dynamic Memory Tools

These tools are essentially wrappers on the native memory allocation functions, with the addition of Toolkit error handling. In C these native functions include *malloc*, *calloc*, and *free*. The purpose of providing these wrappers is to enable the tracking of memory usage in the production environment. (Note that currently there is no SDPS mechanism external to the Toolkit which does this.) In contrast to the shared memory tools, these tools are for use within a single executable of your code. These functions use link-list utilities internally, in order to keep track of memory that has been allocated.

A brief description of each tool follows.

PGS_MEM_Malloc allocates an arbitrary number of bytes in memory.

PGS_MEM_Calloc allocates an arbitrary number of bytes in memory, and initializes them to zero.

PGS_MEM_Zero initializes an arbitrary block of memory to zero.

PGS_MEM_Realloc reallocates an arbitrary number of bytes to a variable which had previously been allocated memory by PGS_MEM_Malloc or PGS_MEM_Calloc.

PGS_MEM_Free deallocates a given block of memory that was previously allocated by PGS_MEM_Malloc, PGS_MEM_Calloc, or PGS_MEM_Realloc.

PGS_MEM_FreeAll deallocates all memory that was previously allocated by PGS_MEM_Malloc, PGS_MEM_Calloc, or PGS_MEM_Realloc within a given executable.

Most of these functions return a pointer *ptr* to memory, which looks like

```
(void **) &ptr
```

in your code, in the argument to the Toolkit function. This form is necessary because the type of the variable is not known to the Toolkit function.

Please note that **all addresses passed to these tools must be initialized first**, if they have previously held allocated memory and were subsequently freed. This is a general requirement on any re-use of pointers. If this is not done, very strange behavior may result. The tool examples explicitly indicate what needs to be done.

5.1.2 Shared Memory Tools

These tools are for sharing memory between executables, within a single PGE.

You might want to share data between executables this way, in order to save the processing time that would ordinarily go to file I/O, if you were writing to and then reading from a file.

5.1.2.1 Preparing your shell script

This section is a step-by-step explanation of how to use shared memory tools.

A PGE may consist of one or more of your executables, bound by a shell script which you write. In order to use shared memory, you must have at least two executables in the PGE. Here we use an example which consists of three executables, one for each of Level 1A, Level 1B, and Level 2 processing. Assume that a block of memory is to be shared among all 3 executables.

A simple PGE shell script which you build for use with the TK4 software might look like this:

```
# File /usr/test/sample_PGE
# Sample PGE shell script

level_1a.exe
level_1b.exe
level_2.exe
```

In order to use shared memory at the SCF, you **must** either call your PGE script from PGS_PC_Shell.sh, like this

```
unix% PGS_PC_Shell.sh /usr/test/sample_PGE 1111
```

or alternatively construct your own script using PGS_PC_InitCom and PGS_PC_TermCom, like this

```
# File /usr/test/sample_SCF_script
# Sample shell script for encapsulating PGS at the SCF

PGS_PC_InitCom 1 1
/usr/test/sample_PGE
PGS_PC_TermCom 1 1
```

Ordinarily you would do it the first way.

What the PC software does is to prepare for the use of shared memory, among other things.

Note that only the PGE script */usr/test/sample_PGE* would be delivered to the DAAC; *PGS_PC_Shell.sh*, *PGS_PC_InitCom* and *PGS_PC_TermCom* are part of the DAAC environment, and are provided as part of the Toolkit only for purposes of your testing at the SCF. You needn't include calls to them in your delivered code. Access to shared memory is automatically available at the DAAC.

5.1.2.2 Using Toolkit shared memory tools in your executables

In the **first** executable in which you use shared memory (here *level_1a.exe*), you must call PGS_MEM_ShmCreate. This tool actually initializes your shared memory block by allocating space for it in system memory. Only one user-specified memory block is allowed per PGE.
Also, this function is to be called only once per PGE. Subsequent calls return an error message.

In **each** executable in which you use shared memory (here *level_1a.exe*, *level_1b.exe* and *level_2.exe*), you must do at least one thing:
Call PGS_MEM_ShmAttach. This function "attaches" the user shared memory block to your process. Essentially this means that the shared memory is now available to you.

If you no longer need the shared memory within the current executable, then you may "detach" it. Do this by calling PGS_MEM_ShmDetach. Any data you wrote to the memory block is still there after this call.
At the end of the current executable, the system automatically detaches the shared memory block anyway, so this call is optional. However, any memory that the shared memory block uses is unavailable as dynamic memory during the current executable. Therefore it is desirable to call PGS_MEM_ShmDetach in order to make this memory available again to your process, if you no longer need the shared memory.

At the end of your PGE, the shared memory is deleted by use (at the SCF) of PGS_PC_Shell.sh or PGS_PC_TermCom.

There are system utilities, callable from the Unix command line, which monitor(*ipcs*) or remove(*ipcrm*) shared memory. However, these should not be used as a substitute for the Toolkit functions, as this will only lead to problems at the DAAC.

Note: Toolkit shared memory functions are not POSIX compliant, since no POSIX standard yet exists for these types of functions. If and when a POSIX standard for memory management is available, we may need to change the calling sequence for the Toolkit shared memory tools. However, this will probably not be necessary.

The allowable maximum amount of shared memory is TBD, because of concerns about machine dependencies.

5.1.2.3 How the Toolkit itself uses shared memory

The lower-level modules of the Toolkit also use shared memory, to support your use of it, and to make the Toolkit itself more efficient. The Toolkit (system) shared memory block is separate from your (user) shared memory block.

These two blocks of shared memory, yours and the Toolkit's, are the only two blocks of shared memory available.

5.1.3 Fortran, Cray and COTS Considerations

ANSI Fortran 77 does not support manipulation of pointer variables, so no Fortran 77 Toolkit Memory Management functions are available. Tools that support memory management in Fortran 90 are currently under study.

The Cray YMPEL which we at ECS used to test tools on, does not support shared memory. Therefore the shared memory tools were not tested on the Cray.

There is a problem with COTS tools that allocate dynamic memory, specifically IMSL. The problem is that there appears to be no way for the Toolkit dynamic memory functions to track or free this memory. We are working on this.

5.2 Memory Management (MEM) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_MEM_* tools.

5.2.1 PGS_MEM_Calloc

Short explanation of what it's for: Allocate memory in your process, initializing it to zero.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes the file with handle *fileHandle* has already been opened.

C example:

```
#include <PGS_MEM.h>
float *data;
int n_items = 10;
int fsize;
PGSt_IO_Gen_FileHandle *fileHandle;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
Initialize data pointer
   (required if you previously freed memory associated with it;
     see Notes)
Allocate memory, initializing it to zero
Read 5 items of data from a file to partially fill array
*/
data = (float *)NULL;
fsize = sizeof(float);
returnStatus =
PGS_MEM_Calloc( (void **) &data, n_items, fsize );
if( returnStatus == PGS_S_SUCCESS )
{
   fread( data, fsize, 5, fileHandle );
}
/*
array data now contains 5 floating point values, as read
from file fileHandle, plus 5 zero values
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

In the example, the line

```
data = (float *)NULL;
```

is **required** before any call to PGS_MEM_Calloc, if you have previously used the same pointer *data* in a call to PGS_MEM_Free or PGS_MEM_FreeAll. **Behavior of your process is unpredictable if this line is not present.** Effort will be made to check this at DAAC Integration and Test.

The only difference between PGS_MEM_Malloc and PGS_MEM_Calloc is that PGS_MEM_Calloc initializes the variable to zero.

5.2.2 PGS_MEM_Free

Short explanation of what it's for: Free memory for a single variable.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes the variable *data* has previously had memory allocated by either PGS_MEM_Malloc or PGS_MEM_Calloc.

C example:

```
#include <PGS_MEM.h>
float *data;
/*
Begin example
*/
/*
Free memory
*/
PGS_MEM_Free( data );
/*
Initialize data pointer
   (required if you want to re-use this pointer in a later call
     to PGS_MEM_Malloc or PGS_MEM_Calloc; see Notes)
*/
data = (float *)NULL;
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

The line

```
data = (float *)NULL;
```

is **required** before any call to PGS_MEM_Malloc or PGS_MEM_Calloc, if you have previously used the same pointer variable *data* in a call to PGS_MEM_Free or PGS_MEM_FreeAll. Therefore it is prudent to do this right after you free the memory, just in case you forget later. **Behavior of your process is unpredictable if this line is not present.**

Use of this function is optional. All dynamically allocated memory is automatically freed to the system at the termination of the current executable.

5.2.3 PGS_MEM_FreeAll

Short explanation of what it's for: Free all memory that you previously allocated dynamically.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes that you have previously allocated some memory using PGS_MEM_Malloc and/or PGS_MEM_Calloc.

C example:

```
#include <PGS_MEM.h>
/*
Begin example
*/
/*
Free all dynamically allocated memory
*/
PGS_MEM_FreeAll();
/*
All dynamically allocated memory within this exectuable has
now been freed
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

After you use this function, if you want to re-use any of the pointers to which you previously dynamically allocated memory, you **must** first re-initialize them to zero, before any call to PGS_MEM_Malloc or PGS_MEM_Calloc. **Behavior of your process is unpredictable if this is not done.**

Use of this function is optional. All dynamically allocated memory is automatically freed to the system at the termination of the current executable.

5.2.4 PGS_MEM_Malloc

Short explanation of what it's for: Allocate memory in your process.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes the file with handle *fileHandle* has already been opened.

C example:

```
#include <PGS_MEM.h>
float *data;
int n_items = 10;
int fsize;
PGSt_IO_Gen_FileHandle *fileHandle;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
/*
Initialize data pointer
   (required if you previously freed memory associated with it;
     see Notes)
Allocate memory
Read 10 items of data from a file to fill array
*/
data = (float *)NULL;
fsize = sizeof(float);
returnStatus =
PGS_MEM_Malloc( (void **) &data, n_items*fsize );
if( returnStatus == PGS_S_SUCCESS )
{
   fread( data, fsize, n_items, fileHandle );
}
/*
array data now contains 10 floating point values, as read
from file fileHandle
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

In the example, the line

```
data = (float *)NULL;
```

is **required** before any call to PGS_MEM_Malloc, if you have previously used the same pointer *data* in a call to PGS_MEM_Free or PGS_MEM_FreeAll. **Behavior of your process is unpredictable if this line is not present.** Effort will be made to check this at DAAC Integration and Test.

The only difference between PGS_MEM_Malloc and PGS_MEM_Calloc is that PGS_MEM_Calloc initializes the variable to zero.

5.2.5 PGS_MEM_Realloc

Short explanation of what it's for: Re-allocate memory in your process, to a variable to which memory has been allocated previously. Useful for extending arrays to a longer length than originally allocated.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes that array variable *data* of dimension 10 has had memory allocated previously by PGS_MEM_Malloc or PGS_MEM_Calloc, as in the examples given in those tool descriptions.

C example:

```
#include <PGS_MEM.h>
float *data;
int n_items = 20;
int fsize;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
fsize = sizeof(float);
returnStatus =
PGS_MEM_Realloc( (void **) &data, n_items*fsize );
/*
Array data now contains space for 20 floating point values
The first 10 values are the same as before; the second 10 values
   contain garbage
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

Only pointer variables which have been used in a previous call to PGS_MEM_Malloc or PGS_MEM_Calloc should be used in a call to PGS_MEM_Realloc. Do **not** use a pointer variable which has been freed and not re-initialized to zero; if you do, behavior of your process will be unpredictable.

5.2.6 PGS_MEM_ShmAttach

Short explanation of what it's for:

Make previously allocated block of shared memory available to your process, so you can put data to it or get data from it.

This function is in file: $PGSSRC/MEM/PGS_MEM1.c

Examples:

This example is a continuation of the one used in PGS_MEM_ShmCreate.

Two examples are presented: in the first, we show how the shared memory is attached in your first executable *level1a.exe*, then filled with data; In the second, we attach it again in your second executable *level1b.exe*, then access the shared memory data. (This is also valid for subsequent executables.)

The examples show one way of how you might share ephemeris data among your executables.

C example 1: First executable *level1a.exe*

Example 1 assumes
(1) you have already processed your data enough (via Toolkit calls to ephemeris tools) to know the total number of ephemeris points *npts*;
(2) you have already called PGS_MEM_ShmCreate.

```
#include <PGS_MEM1.h>

/*
Structure for storing ephemeris data
Memory assumed allocated previously for structure elements
*/
typdef struct
{
   long *clockTime;
   double *xPosition;
   double *yPosition;
   double *zPosition;
} ephemStruct;

ephemStruct *ephemeris;

int lsize;
int dsize;
long size1l;
long size1d;
long totsize;

/*
Previously determined total number of data points in structure
*/
long npts;

/*
Intermediate arrays previously allocated and filled with data
*/
long *clockTime;
double *xPosition;
double *yPosition;
double *zPosition;

PGSt_SMF_status returnStatus;

/*
Begin example
*/

/*
First make the shared memory available to your process
*/

returnStatus = PGS_MEM_ShmAttach( (void **) &ephemeris );

/*
Now copy data from intermediate arrays into shared memory
*/

lsize = sizeof(long);
dsize = sizeof(double);
size1l = npts * lsize;
size1d = npts * dsize;

memcpy( ephemeris->clockTime, clockTime, size1l );
memcpy( ephemeris->xPosition, xPosition, size1d );
memcpy( ephemeris->yPosition, yPosition, size1d );
memcpy( ephemeris->zPosition, zPosition, size1d );

/*
Ephemeris data is now saved in shared memory

To make the shared memory available to another executable
later in the same PGE, call PGS_MEM_ShmAttach again
*/
```

C example 2: Second executable *level1b.exe*

Example 2 assumes that you have already filled the shared memory block with data, as in example 1.

```
#include <PGS_MEM1.h>
#include <math.h>
#define REARTH 6.4E6  /* crude earth radius (m) */
/*
Structure for storing ephemeris data
Memory assumed allocated previously for structure elements
*/
typdef struct
{
   long *clockTime;
   double *xPosition;
   double *yPosition;
   double *zPosition;
} ephemStruct;

ephemStruct *ephemeris;

double altitude;
PGSt_SMF_status returnStatus;

/*
Begin example
*/

/*
Make the shared memory available to your process
*/

returnStatus = PGS_MEM_ShmAttach( (void **) &ephemeris );

/*
Ephemeris data is now available to your process

For example, you might calculate the altitude of the spacecraft:
*/

altitude = sqrt( sqr(ephemeris->xPosition) +
     sqr(ephemeris->yPosition) + sqr(ephemeris->zPosition) )
      - REARTH;
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

This function must be the third (or later) Toolkit shared memory function called in your code. PGS_MEM_ShmSysInit and PGS_MEM_ShmCre ate must have been called first, before calling this function.
You call PGS_MEM_ShmAttach once in each executable where you need to access the shared memory data.

It is preferred that you use one single structure for all of your shared memory. This is how these tools were tested. It is possible that spurious results may be obtained if you use more than one structure, or some other combination of data types.

Do not fill any variables in the shared memory block with data until **after** you call PGS_MEM_ShmAttach. Any data you put in these variables before the call may be overwritten.

The example used in no way reflects how the actual Toolkit ephemeris tools format this data. This is a contrived example for purposes of illustration only.

This function is not POSIX compliant, nor are any Toolkit shared memory functions.

5.2.7 PGS_MEM_ShmCreate

Short explanation of what it's for:

Used to Initiate the use of your shared memory block.

This function is in file: $PGSSRC/MEM/PGS_MEM1.c

Examples:

The example shows one way of how you might share ephemeris data among your executables. (Please note that this in no way reflects how the actual Toolkit ephemeris tools format this data.)

The example assumes
(1) you have already processed your data enough (via Toolkit calls to ephemeris tools) to know the total number of ephemeris points *npts*;
(2) you have already called PGS_MEM_ShmSysInit.

C example:

```
#include <PGS_MEM1.h>

/*
Structure for storing ephemeris data
*/
typdef struct
{
   long *clockTime;
   double *xPosition;
   double *yPosition;
   double *zPosition;
} ephemStruct;

ephemStruct *ephemeris;

int lsize;
int dsize;
long size1l;
long size1d;
long totsize;

/*
Previously determined total number of data points in structure
*/
long npts;

PGSt_SMF_status returnStatus;

/*
Begin example
*/

/*
First allocate memory for and copy over ephemeris data from
   intermediate arrays
*/
lsize = sizeof(long);
dsize = sizeof(double);

ephemeris->clockTime = (long *)NULL;
ephemeris->xPosition = (double *)NULL;
ephemeris->yPosition = (double *)NULL;
ephemeris->zPosition = (double *)NULL;

returnStatus =
PGS_MEM_Calloc( (void **) &(ephemeris->clockTime), npts,lsize);
returnStatus =
PGS_MEM_Calloc( (void **) &(ephemeris->xPosition), npts,dsize);
returnStatus =
PGS_MEM_Calloc( (void **) &(ephemeris->yPosition), npts,dsize);
returnStatus =
PGS_MEM_Calloc( (void **) &(ephemeris->zPosition), npts,dsize);

/*
Calculate total size of shared memory block
Reserve space for your shared memory block
*/

size1l = npts * lsize;
size1d = npts * dsize;
totsize = sizeof( ephemStruct ) + size1l + 3*size1d;
returnStatus = PGS_MEM_ShmCreate( totsize );

/*
Space has now been reserved in shared memory for your structure
You are now ready to call PGS_MEM_ShmAttach to make the shared
memory available to your process
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

This function must be the second Toolkit shared memory function called in your code. You only need to call it once per PGE; subsequent calls are ignored.

PGS_MEM_ShmSysInit must have been called first, before calling this function.

After you call this function, you must call PGS_MEM_ShmAttach to actually make the shared memory available in your code, in each executable where you want to use it.

What this function actually does is reserve a given amount of space in system memory for your shared memory block. You cannot make this amount bigger later; you must reserve it all at once by using this function.

It is preferred that you use one single structure for all of your shared memory. This is how these tools were tested. It is possible that spurious results may be obtained if you use more than one structure, or some other combination of data types.

Do not fill any variables in the shared memory block with data until **after** you call PGS_MEM_ShmAttach. Any data you put in these variables before the call may be overwritten.

This function initializes the use of your shared memory block, in contrast to PGS_MEM_ShmSysInit, which initializes the shared memory block that the Toolkit itself uses.

This function is not POSIX compliant, nor are any Toolkit shared memory functions.

5.2.8 PGS_MEM_ShmDetach

Short explanation of what it's for:

Optionally releases shared memory block, so the current executable can no longer access it.

This function is in file: $PGSSRC/MEM/PGS_MEM1.c

Examples:

C example

```
#include <PGS_MEM1.h>

PGS_MEM_ShmDetach();

/*
Shared memory data is no longer available to this executable

To make the shared memory available to another executable
later in the same PGE, call PGS_MEM_ShmAttach again
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

The shared memory data is not erased by this function, it is simply no longer available to your current executable. To make it available again in this or a subsequent executable in this same PGE, call PGS_MEM_ShmAttach again.

Use of this function is optional. The system automatically detaches the shared memory block at the end of each executable.
However, if you are no longer using the shared memory block in the current executable, it is a good idea to detach it, so that the memory is available as dynamic memory to your process. You may detach and re-attach the shared memory block as much as you like.

This function is not POSIX compliant, nor are any Toolkit shared memory functions.

5.2.9 PGS_MEM_Zero

Short explanation of what it's for: Sets a given amount of memory to zero.

This function is in file: $PGSSRC/MEM/PGS_MEM.c

Examples:

Example assumes that variable *data* has had memory allocated previously by PGS_MEM_Malloc, PGS_MEM_Calloc, or PGS_MEM_Realloc, as in the examples given in those tool descriptions.

C example:

```
#include <PGS_MEM.h>
float *data;
int n_items = 10;
int fsize;
/*
Begin example
*/
fsize = sizeof(float);
PGS_MEM_Zero( data, n_items*fsize );
/*
array data now contains zero in all 10 positions
*/
```

Fortran example:

Dynamic memory allocation is not allowed in Fortran 77. Fortran 90 tools are under study.

Notes:

You might use this function after using PGS_MEM_Realloc, to set the re-allocated memory to zero; or you just might want to reinitialize an array for re-use.

Warning: Be careful not to zero out past the ends of the bounds of an array. This would cause some other variable to be set to zero, giving unpredictable results in your program.

6. Metadata (MET) Tools

6.1 Metadata (MET) Tools Overview

6.1.1 Introduction

This set of tools is designed to manage the metadata inserted into each EOS product; i.e. the per granule metadata. The user is the science PGE which initiates the tools in a specified sequence, to obtain and marshal metadata values.

Within ECS the term "Metadata" relates to all information of a descriptive nature which is associated with the product or dataset. Metadata of importance to production software developers are:

- Data elements commonly found in a product file header, such as temporal or spatial coverage
- documentation that accompanies the production algorithm software
- data origin information

In order to establish standards for the EOS project, a minimal set of parameters has been made mandatory to accompany standard products. This set is detailed in the ECS document DID 311.

Listed below are mandatory metadata parameters:

- Longname collection (dataset) name
- Spatial_coverage (group) one of spatial coverage options
- Temporal_coverage (group) one of temporal coverage options
- UR_OF_ECS_product_input ID of input product used to generate this product
- Quality_rating (group) all of group
- reprocessing_status (group) 2 attributes denoting status

Other Optional parameters:

- size_MB_ECS_data_granule size of product
- UR_of_ancillary_input_granules ID of ancillary product used to generate this product
- UR_of_Orbit_paramters_granule ID of orbit parameters file used to generate this product

Further information describing the attributes can be found in appendix J of the Toolkit Users Guide

The establishment of metadata values for ECS produced products will be important for the services which will be applied to the data upon request by users. This includes, for example, subsetting by geolocation. Note that the term 'users' could mean human or the production system itself.

6.1.2 Accessing Metadata

In order to manage the metadata in the ECS, and to avoid future changes in the toolkit software interface affecting user code, we have designed the metadata access to be file driven. The Toolkit metadata tools will rely on a Metadata Configuration File (MCF). The purpose of the MCF is to provide a structured medium which acts as a repository for the attributes which will be attached to data products. A template for constructing an MCF is provided with Toolkit 5 (Aug. 95), to be edited by the instrument software development teams. The template, as presented in the example consists of two sections or GROUPS, namely granule and product specific. The attributes in the granule GROUP, represent core metadata. The attributes in the product specific GROUP are to be established at the discretion of each instrument team. It is expected that one MCF per data product will be specified.

Note: Any new additions to the MCF, must also be mirrored in the data dictionary.

An example of an MCF is presented below. The MCF is provided to each Instrument Team as a template. The MCF has been designed around the Object Development Language (ODL) libraries, which access data in a Group, Object and Attribute context. Each meaningful collection of data, described by a name is known as an OBJECT. Individual pieces of information about the form and content of the object are called ATTRIBUTES. When it is convenient to group together a number of objects under one label, a GROUP is constructed. For more information on ODL and documentation, please press icon.

blocked URL

ODL enables the metadata tools to access data held within the MCF, and to output values to the MCF. Data are held in a PARAMETER = VALUE (PVL) format. For information on PVL please perform a search using the keyword PVL at the following site, please press icon.

blocked URL

Below is an example of PVL syntax from the Metadata Configuration File (MCF).

GROUP = GRANULEDATAOBJECT = LongNameData_Location = "MCF"Mandatory = "TRUE"Value = "MODIS_SST"END_OBJECT = LongNameEND_GROUP = GRANULEDATA

The parameter, Data Location may be to any of the following:

- Data Location = PCF. Values are obtained automatically from the Process Control File
- Data Location = MCF. Values are preset during the creation of the MCF from the template by the Instrument Team.
- Data Location = PGE. Values are set by the PGE.

If the Data_Location field is filled in with the value "MCF", there will always be a Value = field. With Data_Location = PGE or PCF , there is no value field; this is added when the MCF is written to the product. The Mandatory field denotes whether or not the attribute is mandatory. If it is mandatory and it is not set by the PGE a warning message is returned. Strict adherence to the format and structure of the MCF is advisable, to prevent any needless or spurious errors.

Values written to the MCF while it is held in memory after initialisation are checked against a data dictionary. The data dictionary is supplied with the toolkit routines at present. The data dictionary provided with the toolkit is in PVL/ODL format. It contains data descriptions relevant to all attribute to be found in the granule specific group. Any additional information pertinent to product specific Metadata must be added by the Instrument team. At present, there is no "keeper" of the data dictionary. In the future, the data dictionary as well as the MCF will be prepared by the Data Server.

A scenario for Tool usage can be seen below:

STEP 1 - Initialize MCF

PGS_MET_Init(filelogical, metadata handles)

STEP 2 - Extract Value from file in PCF

PGS_MET_GetPCAttr(product file id, product version number, name of hdf attribute containing metadata, metadata parameter, returned metadata parameter value)

The output will be the value of the metadata attribute from the HDF metadata attribute or header. e.g. Obtain the QA_%_of_MissingData from an input product.

STEP 3 - Write the value extracted to the MCF in memory

PGS_MET_SetAttr(metadata group name, name.class of parameter, value to be inserted)

This will locate the group in the MCF, then the object name, and class if this is specified, and attach a new attribute to the object, which will hold the value to be associated with that attribute. The value will also be checked against the data dictionary; if the value is within the specified range, and of the correct type, it will be associated. (i.e. held in memory location)

STEP 4 - A value already held in the MCF in memory is needed to calculate a new value for a product specific object.

PGS_MET_GetSetAttr(metadata group name, name.class of parameter, value to be passed back)

STEP 5 - In order to calculate this new value, information is also needed from the Configuration parameters set up in the Process Control File.

PGS_MET_GetConfigData(name of parameter, value to be passed back)

This will search the Process control file, and return the value back to the algorithm.

STEP 6 - The PGE has used the two inputs to calculate a new value for one of the MCF objects, and wants to write it to the MCF held in memory.

PGS_MET_SetAttr(metadata group name, name.class of parameter, value to be inserted)

STEP 7 - The PGE has finished setting all the values which are mandatory in the MCF, but there is still some relevant granule information which the PGE wants to add to the MCF. The PGE accomplishes this by adding this information to the PRODUCT_SPECIFIC_METADATA group. Located within this group lie the object names the instrument team has already specified as being product specific.

PGS_MET_SetAttr(product specific metadata group name, name.class of parameter, value to be inserted)

STEP 8 - After multiple calls to PGS_MET_SetAttr the MCF in memory is now complete, all the granule specific metadata have been set, and the relevant product specific metadata have been set. The PGE now writes the metadata out as an HDF attribute attached to the product.

PGS_MET_Write(metadata group to be written out, HDF file attribute name, HDF file ID)

6.2 Metadata (MET) Tool Descriptions

The calling sequences for the PGSTK Metadata Tools can be found in the following sections. In order to utilize the tools to their optimum capacity, they must be called in a specified sequence within the algorithm code; i.e., PGS_MET_Init()once for each physical MCF), then PGS_MET_SetAttr() (0-ntimes), then PGS_MET_Write() (once for each HDF attribute). PGS_MET_GetSetAttr(), PGS_MET_GetPCAttr() and PGS_MET_GetConfig() can be called any number of times at any point after Init and before PGS_MET_Write().

6.2.1 PGS_MET_Init

The first step in reading from or writing to the MCF is with initialization. The contents of the MCF are read into memory and any values which are to be set automatically from the PCF (i.e. where location = PCF), are located and inserted. Any values preset in the MCF(Data_Location = MCF) are checked against the data dictionary. The MCF is also checked to see if it is in the correct ODL syntactical format.

For calling sequences go to PGS_MET_Init.

6.2.2 PGS_MET_SetAttr

The PGE can use the PGS_MET_SetAttr tool to set values already known to the algorithm, or to set those values which are available from the Process Control File (PCF). This function also acts to check out the validity of the value being set. The value is checked against the data dictionary for type and whether it falls within a predefined range.

For calling sequences go to PGS_MET_SetAttr.

6.2.3 PGS_MET_GetSetAttr

When the PGE needs to find and use a value from the MCF after it has been intialized, PGS_MET_GetSetAttr is used. This tool is used to get values pre - set by the Instrument Team, i.e. where data_Location is set to MCF.

For calling sequences go to PGS_MET_GetSetAttr.

6.2.4 PGS_MET_GetPCAttr

The first method which enables the algorithm (PGE) to extract metadata values from the PCF is by using PGS_MET_GetPCAttr. This call retrieves parameters in the PCF which are either located as an HDF attribute on product files, or can be found in a separate ASCII file.

NOTE: These ASCII files must be in flat ODL format. The HDF attributes are guaranteed to be in this format if they have been written out to the file using the PGS_MET_Write function.

For calling sequences go to PGS_MET_GetPCAttr.

6.2.5 PGS_MET_GetConfig

The second method which enables the algorithm (PGE) to extract metadata values from the PCF is by using PGS_MET_GetConfigData. This call enables the user to obtain the configuration data parameters held within the PCF.

For calling sequences go to PGS_MET_GetConfig.

6.2.6 PGS_MET_Write

Once the algorithm (PGE) has finished retrieving and setting all the values in the mandatory section of the MCF, and the specific attributes relevant to that specific product, the final stage is to write the granule and product specific values to the product. PGS_MET_Write writes the values out to an HDF file as an HDF 'attribute'. This tool can write certain groups within the MCF to various locations within the HDF file, e.g. the granule group can be written to the HDF file as a global attribute, and if a product specific group is present in the MCF it may be written as a local attribute.

NOTE: To see examples of the format of the resulting metadata written to the product, consult Appendix J of the Toolkit Users Guide.

For calling sequences go to PGS_MET_Write.

6.2.7 PGS_MET_Remove

This tool frees up the memory allocated by the ODL libraries. The representation of the MCF data dictionary will be removed rom memory.

For calling sequences go to PGS_MET_Remove.

7. Ephemeris and Attitude Data Access (EPH) Tools

7.1 Ephemeris and Attitude Data Access (EPH) Tools Overview

This section describes how to access spacecraft ephemeris and attitude data through the Toolkit.

7.1.1 Introduction

The source of ECS orbit and attitude data is platform dependent. It may include Flight Dynamics Facility (FDF) files and/or Level 0 data from platform ancillary packets in some combination of primary and backup orbit and attitude data. At this writing (11/94), neither simulated Level 0 ephemeris data nor simulated FDF files are available.

In the interim, it is desired to have a means of generating simple ephemeris and attitude data from known orbital elements, for use at the SCF. To this end we have produced an orbit and attitude simulator, which produces files in a format that may be read by functions in the EPH Toolkit group; it works for TRMM, EOS AM and EOS PM platforms. While while these functions currently reads only files generated by the simulator, the intention is that they will read real ephemeris files in the future.

The format and mechanism of how orbit and attitude data become available to the Toolkit at the DAAC is TBD. One possibility is that regardless of source or platform the data will be reformatted to a single format, e.g., the one we have defined for the simulator output. To this end, a new ECS requirement has been generated for preprocessing all ephemeris data to a common format. Whatever the decision on this mechanism, every effort will be made to keep the Toolkit function calling sequence unchanged.

This section of the Primer consists of two parts: (1) description of how to construct a set of simulated ephemeris files for your use at the SCF, and (2) description of the Toolkit function you use in your code to read this file.

For specifics about orbit and attitude data from FDF and platform ancillary packets, see Level 0 Data Issues for the ECS Project, sec. 5.

7.1.2 Preparing a Simulated Ephemeris/Attitude File Set

This section shows how to generate a set of files that the Toolkit ephemeris and attitude tools can read. These files are intended for testing software functionality and not for mission planning.

7.1.2.1 Using the Toolkit simulator to create an ephemeris/attitude file set

The program that generates this file is an interactive one. We give a sample session on how it works.

In the example, *data that you type is given like this*;

`data generated by the program is given like this.`

The line
-->
means that you typed a carriage return, so using the default value.

`unix%` is the Unix system prompt.

SAMPLE SESSION:

`unix%` *$PGSBIN/orbsim*

```
       ********************
       * ------O--------- *
       * ___/\__/\_______ *
       * __/  \/  \______ *
       *  /    \___\_____ *
       * /                *
       * ^^^^^^^^^^^^^^^^ *
       * ^^^^^^^^^^^^^^^^ *
       * ^^^^^^^^^^^^^^^^ *
       * =EOS=            *
       ********************

  ECS SPACECRAFT ORBIT SIMULATOR

Enter <return> at a prompt to select the default
option (indicated by []).  Enter 'q' at any prompt
to quit.

enter spacecraft ID (TRMM, EOS_AM, EOS_PM):
```

-->*TRMM*

```
enter start and stop dates in CCSDS ASCII (format A or B)
 A) YYYY-MM-DD
 B) YYYY-DDD

enter start date:
```

-->*1998-06-30*

```
enter stop date [1998-06-30]:
-->
enter time interval in seconds [ 60.000000 sec]:
-->

start day:    1998-06-30
stop day:     1998-06-30
time interval: 60.000000 seconds

This will create approximately 0.18 MB of data.
accept ([y]/n)?
-->

You may introduce random noise in the attitude and attitude
rates.  This noise will be deviations from the nominal values
of zero for these quantities (i.e. spacecraft reference frame
identical to orbital reference frame).  The number entered will
be the maximum deviation (+/-) from the nominal values.  The same
value will be used for all the components of a particular
quantity.  Enter 'N' at the first prompt for no noise at all.
Otherwise entering zero for a particular quantity will preclude
noise in that state.  The default case is no noise.
Enter noise level in arcseconds (0.00-999.99).

enter attitude noise level [N]:
```

-->*50*

```
enter attitude rate noise level [50.00]:
-->

By default this program will install the files it generates in: /pgs_home/lib/database/EPH.

install files in [/pgs_home/lib/database/EPH]:
-->

creating file: /pgs_home/lib/database/EPH/TRMM_1998-06-30.eph
creating file: /pgs_home/lib/database/EPH/TRMM_1998-06-30.att

Done.  Generate another set of ephemeris files (y/[n]):
-->

unix%
```

The result is that you now have a set of files named:

```
/pgs_home/lib/database/EPH/TRMM_1998-06-30.att
/pgs_home/lib/database/EPH/TRMM_1998-06-30.eph
```

where *pgs_home* is the directory where you installed the Toolkit. These files contains ephemeris and attitude data (respectively) for the TRMM spacecraft, for June 30, 1998, spaced at intervals of 60.0 sec, with attitude noise amplitude of 50 arcsec and attitude rate noise amplitude 50 arcsec /sec. The total size of the files is about 0.18 MB.

If you had put a different stop date, the simulator would have created a separate set of ephemeris/attitude files for each day requested.

7.1.2.2 Creating your own ephemeris and attitude data file

You may also create your own files. They must be in the same format as the simulator output for the Toolkit functions to read them.

There is a Toolkit utility you can use, that checks that the files you construct are consistent with the formats the Toolkit ephemeris and attitude access tools expect as input. It is called *chkeph*. To run it, just give it one or more filenames:

unix% *cd /pgs_home/lib/database/EPH*
unix% *$PGSBIN/chkeph TRMM_AM_1998-06-30.att TRMM_AM_1998-06-30.eph*

```
TRMM_1998-06-30.att:
   spacecraft ID: 4444 (TRMM)
   start time: 173318405.000000 (1998-06-30T00:00:00.000000)
   stop time: 173404745.000000 (1998-06-30T23:59:00.000000)
   time interval: 60.000000
   total records: 1440
   checking record: 0001440 ... OK.

TRMM_1998-06-30.eph:
   spacecraft ID: 4444 (TRMM)
   start time: 173318405.000000 (1998-06-30T00:00:00.000000)
   stop time: 173404745.000000 (1998-06-30T23:59:00.000000)
   time interval: 60.000000
   total records: 1440
   checking record: 0001440 ... OK.
```

7.1.2.3 Adding ephemeris and attitude data file sets to the PCF

In the SCF environment users must populate the PCF with appropriate ephemeris and attitude data files themselves. No tools that require access to spacecraft ephemeris data will function without these ephemeris and attitude files. An ephemeris file and an attitude file must be provided for any time during which processing will be requested.

The PCF file provided with the Toolkit contains the Logical IDs which have been reserved for the ephemeris and attitude data files. There is one Logical ID for each type of data and the appropriate Logical ID MUST be used for each set of ephemeris and attitude files. Replace the dummy values in the PCF with the actual location of the ephemeris and attitude files to be used. Use the given ephemeris file Logical ID for all ephemeris data files and the given attitude file Logical ID for all attitude files. To include multiple files of either type use file versioning. The order of the files is not important, the ephemeris and attitude access tool will sort the files before attempting to access them (WARNING: providing files with overlapping start /stop times may produce unexpected results).

The unconfigured ephemeris and attitude Logical ID entries in the PCF look as follows (respectively):

```
10501|INSERT_EPHEMERIS_FILES_HERE|||||1
10502|INSERT_ATTITUDE_FILES_HERE|||||1
```

The configured entries should look something like this:

```
10501|TRMM_1998-06-30.eph|~/database/sun5/EPH||||1
10502|TRMM_1998-06-30.att|~/database/sun5/EPH||||1
```

When including multiple ephemeris/attitude data sets, use file *versioning*:

```
10501|TRMM_1998-06-30.eph|~/database/sun5/EPH||||3
10501|TRMM_1998-07-01.eph|~/database/sun5/EPH||||2
10501|TRMM_1998-07-02.eph|~/database/sun5/EPH||||1
10502|TRMM_1998-06-30.att|~/database/sun5/EPH||||3
10502|TRMM_1998-07-01.att|~/database/sun5/EPH||||2
10502|TRMM_1998-07-02.att|~/database/sun5/EPH||||1
```

See sec. 4.1.2, Constructing your Process Control file, for information about PCF entries.

7.2 Ephemeris and Attitude Data Access (EPH) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_EPH_* tools.

7.2.1 PGS_EPH_EphemAttit

Short explanation of what it's for: Get spacecraft ephemeris (ECI position and velocity) and attitude (Euler angles, rates, and spacecraft to ECI quaternion) from ephemeris/attitude file.

This function is in file: $PGSSRC/EPH/PGS_EPH_EphemAttit.c

Examples:

Examples get both ephemeris and attitude for 2 times, at a one second interval.

C example:

```
#include <PGS_EPH.h>

PGSt_tag        scTag;
PGSt_integer    numValues;
char            asciiUTC_A_start[28];
PGSt_double     time_offset[2];
PGSt_boolean    get_ephemeris_flag;
PGSt_boolean    get_attitude_flag;
PGSt_integer    qualityFlags[2][2];
PGSt_double     positionECI[2][3];
PGSt_double     velocityECI[2][3];
PGSt_double     ypr[2][3];
PGSt_double     yprRate[2][3];
PGSt_double     attitQuat[2][4];
PGSt_SMF_status returnStatus;

/*
Begin example
*/

scTag = PGSd_TRMM;   /* PGSd_EOS_AM, PGSd_EOS_PM also allowed */
numValues = 2;
strcpy( asciiUTC_A_start, "1998-06-30T10:51:28.320000Z" );

time_offset[0] = 0.0;  /* 1998-06-30T10:51:28.320000Z */
time_offset[1] = 1.0;  /* 1998-06-30T10:51:29.320000Z */

get_ephemeris_flag=PGS_TRUE;/*PGS_FALSE if don't want ephemeris*/
get_attitude_flag=PGS_TRUE; /*PGS_FALSE if don't want attitude*/


returnStatus = PGS_EPH_EphemAttit(
                         scTag, numValues, asciiUTC_A_start, time_offset,
                         get_ephemeris_flag, get_attitude_flag,qualityFlags,
                         positionECI, velocityECI, ypr, yprRate,
                         attitQuat );

/*
Results: the following variables now are filled:

asciiUTC_A_output[0]  "1998-06-30T10:51:28.320000Z" 1st UTC time

positionECI[0][0]  1413531.574 ECI x position (m)
positionECI[0][1] -6005427.214 ECI y position (m)
positionECI[0][2] -2693615.671 ECI z position (m)

velocityECI[0][0]  7005.698  ECI x velocity (m/s)
velocityECI[0][1]  232.091   ECI y velocity (m/s)
velocityECI[0][2]  3166.378  ECI z velocity (m/s)

ypr[0][0] -0.001519 1st Euler angle (rad)
ypr[0][1]  0.000580 2nd Euler angle (rad)
ypr[0][2] -0.005627 3rd Euler angle (rad)
```

```
yprRate[0][0]  0.000967 1st Euler angle rate (rad/s)
yprRate[0][1]  0.009510 2nd Euler angle rate (rad/s)
yprRate[0][2]  0.001334 3rd Euler angle rate (rad/s)

attitQuat[0][0]  0.830083 1st component attitude quaternion
attitQuat[0][1] -0.516056 2nd component attitude quaternion
attitQuat[0][2] -0.186537 3rd component attitude quaternion
attitQuat[0][3] -0.099258 4th component attitude quaternion


asciiUTC_A_output[1]  "1998-06-30T10:51:29.320000Z" 2nd UTC time

positionECI[1][0]  1420536.347 ECI x position (m)
positionECI[1][1] -6005191.205 ECI y position (m)
positionECI[1][2] -2690447.532 ECI z position (m)

velocityECI[1][0]  7003.845 ECI x velocity (m/s)
velocityECI[1][1]   239.928 ECI y velocity (m/s)
velocityECI[1][2]  3169.900 ECI z velocity (m/s)

ypr[1][0] -0.001597 1st Euler angle (rad)
ypr[1][1]  0.000502 2nd Euler angle (rad)
ypr[1][2] -0.005705 3rd Euler angle (rad)

yprRate[1][0]  0.001006 1st Euler angle rate (rad/s)
yprRate[1][1]  0.009549 2nd Euler angle rate (rad/s)
yprRate[1][2] -0.005705 3rd Euler angle rate (rad/s)

attitQuat[1][0]  0.829945 1st component attitude quaternion
attitQuat[1][1] -0.516141 2nd component attitude quaternion
attitQuat[1][2] -0.187061 3rd component attitude quaternion
attitQuat[1][3] -0.098985 4th component attitude quaternion
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_EPH_5.f'
      INCLUDE 'PGS_MEM_7.f'
      INCLUDE 'PGS_SMF.f'

      INTEGER      pgs_eph_ephemattit
      INTEGER      sctag
      INTEGER      numvalues
      CHARACTER*27 asciiutc_a_start
      DOUBLE       time_offset(2)
      INTEGER      get_ephemeris_flag
      INTEGER      get_attitude_flag
      INTEGER      qualityflags(2)(2)
      DOUBLE       positioneci(2,3)
      DOUBLE       velocityeci(2,3)
      DOUBLE       ypr(2,3)
      DOUBLE       yprrate(2,3)
      DOUBLE       attitquat(2,4)

      INTEGER      returnstatus

!
! Begin example
!

      scTag = PGSd_TRMM   ! PGSd_EOS_AM, PGSd_EOS_PM also allowed
      numvalues = 2

      asciiUTC_A_start = '1998-06-30T10:51:28.320000Z'

      time_offset(1) = 0.0;  ! 1998-06-30T10:51:28.320000Z
      time_offset(2) = 1.0;  ! 1998-06-30T10:51:29.320000Z

      get_ephemeris_flag=PGS_TRUE !PGS_FALSE if don't want eph
      get_attitude_flag=PGS_TRUE  !PGS_FALSE if don't want att


      returnStatus = pgs_eph_ephemattit(
                         sctag, numvalues, asciiutc_a_start, utc_offset,
                         get_ephemeris_flag, get_attitude_flag,
                         qualityflags, positioneci, velocityeci,
                         ypr, yprrate, attitquat )
```

```
! Results: the following variables now are filled:

! asciiutc_a_output(1) '1998-06-30T10:51:28.320000Z' 1st UTC time

! positioneci(1)(1)  1413531.574 ECI x position (m)
! positioneci(1)(2) -6005427.214 ECI y position (m)
! positioneci(1)(3) -2693615.671 ECI z position (m)

! velocityeci(1)(1)  7005.698  ECI x velocity (m/s)
! velocityeci(1)(2)  232.091   ECI y velocity (m/s)
! velocityeci(1)(3)  3166.378  ECI z velocity (m/s)

! ypr(1)(1) -0.001519 1st Euler angle (rad)
! ypr(1)(2)  0.000580 2nd Euler angle (rad)
! ypr(1)(3) -0.005627 3rd Euler angle (rad)

! yprrate(1)(1)  0.000967 1st Euler angle rate (rad/s)
! yprrate(1)(2)  0.009510 2nd Euler angle rate (rad/s)
! yprrate(1)(3)  0.001334 3rd Euler angle rate (rad/s)

! attitquat(1)(1)  0.830083 1st component attitude quaternion
! attitquat(1)(2) -0.516056 2nd component attitude quaternion
! attitquat(1)(3) -0.186537 3rd component attitude quaternion
! attitquat(1)(4) -0.099258 4th component attitude quaternion

! asciiutc_a_output(1) '1998-06-30T10:51:29.320000Z' 2nd UTC time

! positioneci(2)(1)  1420536.347 ECI x position (m)
! positioneci(2)(2) -6005191.205 ECI y position (m)
! positioneci(2)(3) -2690447.532 ECI z position (m)

! velocityeci(2)(1)  7003.845 ECI x velocity (m/s)
! velocityeci(2)(2)   239.928 ECI y velocity (m/s)
! velocityeci(2)(3)  3169.900 ECI z velocity (m/s)

! ypr(2)(1) -0.001597 1st Euler angle (rad)
! ypr(2)(2)  0.000502 2nd Euler angle (rad)
! ypr(2)(3) -0.005705 3rd Euler angle (rad)

! yprrate(2)(1)  0.001006 1st Euler angle rate (rad/s)
! yprrate(2)(2)  0.009549 2nd Euler angle rate (rad/s)
! yprrate(2)(3) -0.005705 3rd Euler angle rate (rad/s)

! attitquat(2)(1)  0.829945 1st component attitude quaternion
! attitquat(2)(2) -0.516141 2nd component attitude quaternion
! attitquat(2)(3) -0.187061 3rd component attitude quaternion
! attitquat(2)(4) -0.098985 4th component attitude quaternion
```

Notes:

The 3 output Euler angles correspond to yaw, pitch and roll; the order of these values in the *ypr* and *yprRate* output is platform dependent.

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files must be defined in the Process Control File (PCF).

The PCF template supplied with the Toolkit, *$PGSRUN/$BRAND/PCF.relA* contains the reference for the leap seconds file, if you are using a PCF derived from that template, you need not do anything extra, to enable access to that file.

To access spacecraft ephemeris files in the SCF environment, you must add the appropriate files to the PCF. These files must be created by you for testing purposes at the SCF. Spacecraft ephemeris files must be in the ECS ephemeris file format. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).

See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

7.2.2 PGS_EPH_GetEphMet

Short explanation of what it's for: Get metadata associated with spacecraft ephemeris data.

This function is in file: $PGSSRC/EPH/PGS_EPH_GetEphMet.c

Examples:

Examples retrieve orbit metadata for a 100 minute time span.

C example:

```
#include <PGS_EPH.h>

#define MAX_ORBITS 5   /* maximum number of orbits expected */
#define NUM_POINTS 100 /* number of ephemeris data points */

PGSt_double  offsets[NUM_POINTS];
PGSt_double  orbitDownLongitude[MAX_ORBITS];

PGSt_integer numOrbits;

char         asciiUTC[28];
char         orbitAscendTime[MAX_ORBITS][28];
char         orbitDescendTime[MAX_ORBITS][28];

/* initialize asciiUTC and offsets array with the times for
   actual ephemeris records that will be processed (i.e. by
   some other tool) */

strcpy(asciiUTC,"1998-02-03T19:23:45.123");
for (i=0;i<NUM_POINTS;i++)
{
    offsets[i] = (PGSt_double) i*60.0;
}

/* get the ephemeris metadata associated with these times */

returnStatus = PGS_EPH_GetEphMet(PGSd_EOS_AM,NUM_POINTS,asciiUTC,
                                 offsets,&numOrbits,orbitAscendTime,
                                 orbitDescendTime,orbitDownLongitude);

if (returnStatus != PGS_S_SUCCESS)
{
            :
 ** do some error handling ***
            :
}

/* numOrbits will now contain the number of orbits spanned by the
   data set (as defined by asciiUTC and EPHEM_ARRAY_SIZE offsets).
   orbitAscendTime will contain numOrbits ASCII UTC times
   representing the time of northward equator crossing of the
   spacecraft for each respective orbit.  orbitDescendTime will
   similarly contain the southward equator crossing times and
   orbitDownLongitude will contain the southward equator crossing
   longitudes */
```

FORTRAN example:

```
      implicit none

      include 'PGS_EPH_5.f'
      include 'PGS_TD.f'
      include 'PGS_TD_3.f'
      include 'PGS_SMF.f'

      integer max_orbits/5/   ! maximum number of orbits expected
      integer num_points/100/ ! number of ephemeris data points

      double precision offsets(num_points)
      double precision orbitdownlongitude(max_orbits)

      integer          numorbits

      character*27     asciiutc
      character*27     orbitascendtime(max_orbits)
      character*27     orbitdescendtime(max_orbits)

!  initialize asciiutc and offsets array with the times for
!  actual ephemeris records that will be processed (i.e. by
!  some other tool)

      asciiutc = '1998-02-03t19:23:45.123'
      do 100 i=1,ephem_array_size
          offsets(i) = i*60.D0
 100  continue

!  get the ephemeris metadata associated with these times

      returnStatus = pgs_eph_getephmet(pgsd_eos_am,ephem_array_size,
     >                                 asciiutc,offsets,numorbits,
     >                                 orbitascendtime,
     >                                 orbitdescendtime,
     >                                 orbitdownlongitude)

      if (returnStatus .ne. pgs_s_success) then
                  :
       ** do some error handling ***
                  :
      endif

!  numOrbits will now contain the number of orbits spanned by the
!  data set (as defined by asciiUTC and EPHEM_ARRAY_SIZE offsets).
!  orbitAscendTime will contain numOrbits ASCII UTC times
!  representing the time of northward equator crossing of the
!  spacecraft for each respective orbit.  orbitDescendTime will
!  similarly contain the southward equator crossing times and
!  orbitDownLongitude will contain the southward equator crossing
!  longitudes
```

Notes:

This function will determine the time span of the data set from the input UTC reference time and the offsets array. It will then attempt to retrieve the metadata for all orbits spanned by the data set.

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris

The physical references to these files must be defined in the Process Control File (PCF).

The PCF template supplied with the Toolkit, *$PGSRUN/$BRAND/PCF.relA* contains the reference for the leap seconds file, if you are using a PCF derived from that template, you need not do anything extra, to enable access to that file.

To access spacecraft ephemeris files in the SCF environment, you must add the appropriate files to the PCF. These files must be created by you for testing purposes at the SCF. Spacecraft ephemeris files must be in the ECS ephemeris file format. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).

See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

8. Time/Date (TD) Tools

8.1 Overview

8.1.1 Introduction

This section explains the use of the Time/Date (TD) tools. These are used to do conversions between various time scales and formats. The tools do not get any times themselves; they only translate times that you supply.

Many Toolkit functions use these tools internally.

(Spacecraft clock times are obtained through use of the Level 0 (PGS_IO_L0_*) tools.)

8.1.2 Definition of Time Scales and Formats Used

In this section we give short definitions of the time scales used. Also given are the formats used in the *Toolkit*, and the *Variable name* used in the examples.

GAST: Greenwich Apparent Sidereal timeGAST = Greenwich Mean Sidereal Time + (nutation in longitude)*cos(MEAN obliquity of the ecliptic)Toolkit: hour angle of the true vernal equinox of date at the Greenwich meridian in radiansVariable name: gastGPS: Global Positioning System timeTime broadcast by GPS satellites. Continuous seconds since Jan. 6, 1980 midnight (UTC).Toolkit: Double precision seconds since Jan. 6, 1980 midnight (UTC).Variable name: secGPSSCtime: Spacecraft Clock timeTime recorded by spacecraft clock, and returned in Level 0 data in 8-byte packed CCSDS binary format.Epoch: For EOS AM and PM, 1/1/1958, midnight UTC; for TRMM, 1/1/1993, midnight UTC.Exact format and clock resolution are also platform dependent.Toolkit: 8-byte packed CCSDS binary format.Variable name: scTimeTAI: International Atomic TimeTime derived from atomic measurements. Unit is the SI second.Toolkit: Double precision seconds since Jan. 1, 1993 midnight (UTC).Variable name: secTAI93TDB: Barycentric Dynamical TimeUsed as time scale for ephemerides referred to solar system barycenter. Differs from TDT only by periodic variations, never exceeding +/- 2 milliseconds.Toolkit: Vector of two double precision numbers:1st element: Half-integral Julian date2nd element: Fraction of Julian dateJulian date is days since Jan. 1, 4713 BC, Greenwich Mean Noon.A two element vector is used to allow maximum precision.Adding the two components gives the full Julian date.Variable name: jedTDB[2]TDT: Terrestrial Dynamical TimeUsed as time scale for observations in the near earth environment. Always 32.184 sec larger than TAI.Toolkit: Vector of two double precision numbers:1st element: Half-integral Julian date2nd element: Fraction of Julian dateJulian date is days since Jan. 1, 4713 BC, Greenwich Mean Noon.A two element vector is used to allow maximum precision. Adding the two components gives the full Julian date.Variable name: jedTDT[2]UT1: Universal Time (seconds and Julian Date)A measure of time that conforms on the average to the mean diurnal motion of the sun. Time is counted from 0hrs Greenwich Apparent Solar Midnight. UT1 represents the Earth's axial rotation at the value of one day (86,400 seconds) per full revolution. The actual time unit therefore varies with the Earth's rotational speed. Currently, the mean rate of UT1 is about 0.999999974 the rate of TAI - in other words, the UT1 second is a bit longer than the SI second. This time is offered in two forms: seconds since midnight and Julian Date.Toolkit- (sec): Double precision seconds since midnightVariable name: secUT1Toolkit - (Julian Date): Vector of two double precision numbersVariable Name: jdUT1[2]UTC: Coordinated Universal TimeThe basis of most radio broadcast and legal time systems. Differs from TAI by an integral number of seconds, currently -30 sec (10/95). Difference changes on the order of 1 second per year. Maintained within +/- 0.9 sec of UT1 by use of leap seconds.Toolkit: Two CCSDS ASCII Time Code formats:Format A: 27-character string of the form yyyy-mm-ddThh:mm:ss.ffffffZFormat B: 25-character string of the form yyyy-dddThh:mm:ss.ffffffZ(the trailing "Z" is optional for input values.)Some Toolkit functions allow arrays of offsets from a UTC time to be passed in. In this case, in addition to the UTC value, an array of numbers (C: PGSt_double, FORTRAN: double precision), each of which is an offset in seconds from the UTC time, is passed into the function. Variable names: asciiUTC_A, asciiUTC_B, time_offset

Reference for Consultative Committee for Space Data Systems (CCSDS) time code formats is *Time Code Formats*, CCSDS 301.0-B-2, Blue Book Issue 2, April 1990, CCSDS Secretariat, Communications and Data Systems Division (Code-OS), NASA, Washington DC, 20546

Reference for spacecraft clock time information is the white paper Level 0 Data Issues for the ECS Project, sec. 4.2.8.

Reference for all other time scales is *The Astronomical Almanac for the Year 1994,* U.S. Naval Observatory, U.S. Government Printing Office, Washington DC, 1993.

8.1.3 Time/Date Conversion Matrix

Here we present a matrix for determining which Toolkit function you need to use to perform a given time conversion.

The top of the matrix is the time scale you want to convert **FROM;** the right side of the matrix is the time scale you want to convert **TO.** Matrix entries are the names of Toolkit Time/Date functions, with the "PGS_TD_" prefix omitted.

Time/Date Conversion Matrix for PGS_TD_* Tools

CONVERT FROM TIME SCALE

TO TIME SCALE	UTC-A	UTC-B	TAI	SCtime	GPS
UTC-A	x	ASCIItime _BtoA	TAItoUTC	SCtime_ to_UTC	GPStoUTC
UTC-B	ASCIItime _AtoB	x	-	-	-
TAI	UTCtoTAI	UTCtoTAI	Time Interval	-	-
SCtime	UTC_to_ SCtime	UTC_to_ SCtime	-	x	-
GPS	UTCtoGPS	UTCtoGPS	-	-	x
TDBjed	UTCto TDBjed	UTCto TDBjed	-	-	-
TDTjed	UTCto TDTjed	UTCto TDTjed	-	-	-
UT1	UTCtoUT1	UTCtoUT1	-	-	-
UT1jd	UTCto UT1jd	UTCto UT1jd	-	-	-

For example, to convert from spacecraft clock time to CCSDS UTC Time Code A format, use tool PGS_TD_SCtime_to_UTC.

These functions translate between two different time scales or formats. Function PGS_TD_TimeInterval is an exception, in that it returns the time interval in seconds between two TAI times.

Descriptions of the tools are found in the next section.

8.2 Time/Date (TD) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_TD_* tools.

8.2.1 PGS_TD_ASCIItimeAtoB

Short explanation of what it's for:

Convert CCSDS ASCII Time Code A format to CCSDS ASCII Time Code B format.

This function is in file: $PGSSRC/TD/PGS_TD_ASCIItimeAtoB.c

Examples:

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
char asciiUTC_B[26];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
strcpy(asciiUTC_A,"1998-06-30T10:51:28.320000Z");
returnStatus = PGS_TD_ASCIItime_AtoB(asciiUTC_A,asciiUTC_B);
/*
variable asciiUTC_B now contains the string
"1998-181T10:51:28.320000Z"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_asciitime_atob
      CHARACTER*27 asciiutc_a
      CHARACTER*25 asciiutc_b
      INTEGER returnstatus
C
C Begin example
C
      asciiutc_a = '1998-06-30T10:51:28.320000'
      returnstatus = pgs_td_asciitime_atob(asciiutc_a,asciiutc_b)
C
C variable asciiutc_b now contains the string
C '1998-181T10:51:28.320000Z'
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.2 PGS_TD_ASCIItimeBtoA

Short explanation of what it's for:

Convert CCSDS ASCII Time Code B format to CCSDS ASCII Time Code A format.

This function is in file: $PGSSRC/TD/PGS_TD_ASCIItimeBtoA.c

Examples:

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
char asciiUTC_B[26];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
strcpy(asciiUTC_B,"1998-181T10:51:28.320000Z");
returnStatus = PGS_TD_ASCIItime_BtoA(asciiUTC_B,asciiUTC_A);
/*
variable asciiUTC_A now contains the string
"1998-06-30T10:51:28.320000Z"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_asciitime_btoa
      CHARACTER*27 asciiutc_a
      CHARACTER*25 asciiutc_b
      INTEGER returnstatus
C
C Begin example
C
      asciiutc_b = '1998-181T10:51:28.320000'
      returnstatus =pgs_td_asciitime_btoa(asciiutc_b,asciiutc_a)
C
C variable asciiutc_a now contains the string
C '1998-06-30T10:51:28.320000Z'
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.3 PGS_TD_GPStoUTC

Short explanation of what it's for:

Convert GPS seconds to UTC (CCSDS ASCII Time Code A format).

This function is in file: $PGSSRC/TD/PGS_TD_GPStoUTC.c

Examples:

C example:

```
#include <PGS_TD.h>
PGSt_double secGPS;
char asciiUTC_A[28];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
secGPS = 583239101.320000;
returnStatus = PGS_TD_GPStoUTC(secGPS,asciiUTC_A);
/*
variable asciiUTC_A now contains the string
"1998-06-30T10:51:28.320000Z"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_gpstoutc
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION secgps
      INTEGER returnstatus
C
C Begin example
C
      secgps = 583239101.320000
      returnstatus =pgs_td_gpstoutc(secgps,asciiutc_a)
C
C variable asciiutc_a now contains the string
C '1998-06-30T10:51:28.320000Z'
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.4 PGS_TD_SCtime_to_UTC

Short explanation of what it's for: Convert Spacecraft clock time to UTC (CCSDS ASCII Time Code A format). If input is an array of SCtimes, values are returned as offsets to the first time in the array.

This function is in file: $PGSSRC/TD/PGS_TD_SCtime_to_UTC.c

Examples:

Examples given are for the EOS AM platform. They apply also to the TRMM and EOS PM platforms.

C example:

```
#include <PGS_TD.h>
PGSt_tag spacecraftID;
PGSt_scTime scTime[3][8];
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[3];
PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Spacecraft clock time array must have been already prepared */
/* See notes to see how to retrieve this from L0 data */
/* In this example, we assume that the variable scTime
      contains 8-byte packed CCSDS binary values that
      correspond to the 3 UTC times
           1998-06-30T10:51:28.320000Z
           1998-06-30T10:51:29.320000Z
           1998-06-30T10:51:30.320000Z */

spacecraftID = EOS_AM;         /* or TRMM or EOS_PM */
numValues = 3;

returnStatus = PGS_TD_SCTime_to_UTC( spacecraftID,
             scTime, numValues, asciiUTC_A, time_offset );
/*
variable asciiUTC_A now contains the string
"1998-06-30T10:51:28.320000Z"
array time_offset now contains the values
time_offset[0] = 0.000000
time_offset[1] = 1.000000
time_offset[2] = 2.000000
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_sctime_to_utc
      INTEGER spacecraftid
      CHARACTER*8 sctime(3)
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(3)
      INTEGER returnstatus
C
C Begin example
C
C Spacecraft clock time array must have been already prepared
C See notes to see how to retrieve this from L0 data
C In this example, we assume that the 3-element array scTime
C     contains 8-byte packed CCSDS binary values that
C     correspond to the 3 UTC times
C          1998-06-30T10:51:28.320000Z
C          1998-06-30T10:51:29.320000Z
C          1998-06-30T10:51:30.320000Z
C

      spacecraftid = EOS_AM        ! or TRMM or EOS_PM
      numvalues = 3

      returnstatus = pgs_td_sctime_to_utc( spacecraftid,
     .        sctime, numvalues, asciiutc_a, time_offset )

C variable asciiutc_a now contains the string
C "1998-06-30T10:51:28.320000Z"
C array time_offset now contains the values
C time_offset(1) = 0.000000
C time_offset(2) = 1.000000
C time_offset(3) = 2.000000
C
```

Notes:

Spacecraft clock time as input to this tool is the same format as returned by the Toolkit Level 0 Access tools (available in a future Toolkit delivery). Examples assume that the array *scTime* has previously been retrieved from Level 0 data.

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.5 PGS_TD_TAItoGAST

Short explanation of what it's for:

Convert TAI seconds from Jan 1, 1993 to Greenwich Apparent Sidereal Time (GAST).

This function is in file: $PGSSRC/TD/PGS_TD_TAItoGAST.c

Examples:

C example:

```
#include <PGS_TD.h>
PGSt_double secTAI93;
PGSt_double gast;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
secTAI93 = 173357493.320000;
returnStatus = PGS_TD_TAItoGAST(secTAI93,&gast);
/*
variable gast now contains the value
1.416733538965 which is GAST in radians
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INTEGER pgs_td_taitogast
      DOUBLE PRECISION gast
      DOUBLE PRECISION sectai93
      INTEGER returnstatus
C
C Begin example
C
      sectai93 = 173357493.320000
      returnstatus =pgs_td_taitogast(sectai93,gast)
C
C variable gast now contains the value
C 1.416733538965 which is GAST in radians
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.6 PGS_TD_TAItoUTC

Short explanation of what it's for:

Convert TAI seconds from Jan 1, 1993 to UTC (CCSDS ASCII Time Code A format).

This function is in file: $PGSSRC/TD/PGS_TD_TAItoUTC.c

Examples:

C example:

```
#include <PGS_TD.h>
PGSt_double secTAI93;
char asciiUTC_A[28];
PGSt_SMF_status returnStatus;
/*
Begin example
*/
secTAI93 = 173357493.320000;
returnStatus = PGS_TD_TAItoUTC(secTAI93,asciiUTC_A);
/*
variable asciiUTC_A now contains the string
"1998-06-30T10:51:28.320000Z"
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_taitoutc
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION sectai93
      INTEGER returnstatus
C
C Begin example
C
      sectai93 = 173357493.320000
      returnstatus =pgs_td_taitoutc(sectai93,asciiutc_a)
C
C variable asciiutc_a now contains the string
C '1998-06-30T10:51:28.320000Z'
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.7 PGS_TD_TimeInterval

Short explanation of what it's for: Find the interval between two TAI times.

This function is in file: $PGSSRC/TD/PGS_TD_TimeInterval.c

Examples:

C example:

```
#include <PGS_TD.h>
PGSt_double secTAI93_1;
PGSt_double secTAI93_2;
PGSt_double delta_TAI;
PGSt_SMF_status returnStatus;
/*
Begin example
*/
secTAI93_1 = 173357493.320000; /* 1998-06-30T10:51:28.320000Z */
secTAI93_2 = 173357496.320000; /* 1998-06-30T10:51:31.320000Z */
returnStatus = PGS_TD_TimeInterval(secTAI93_1,secTAI93_2,
                      delta_TAI);
/*
variable delta_TAI now contains the value
3.000000
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_timeinterval
      DOUBLE PRECISION sectai93_1
      DOUBLE PRECISION sectai93_2
      DOUBLE PRECISION delta_tai
      INTEGER returnstatus
C
C Begin example
C
      sectai931 = 173357493.320000; ! 1998-06-30T10:51:28.320000Z
      sectai932 = 173357496.320000; ! 1998-06-30T10:51:31.320000Z
      returnstatus =pgs_td_timeinterval(sectai93_1,sectai93_2,
     .                  &delta_tai)
C
C variable delta_tai now contains the value
C 3.000000
```

Notes:

This function simply subtracts one TAI time from another. Since TAI is a continuous time stream, there are no considerations regarding leap seconds.

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.8 PGS_TD_UTC_to_SCtime

Short explanation of what it's for: Convert UTC time to spacecraft clock time.

This function is in file: $PGSSRC/TD/PGS_TD_UTC_to_SCtime.c

Examples:

Examples given are for the EOS AM platform. They apply also to the TRMM and EOS PM platforms.

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_TD.h>
PGSt_tag spacecraftID;
char asciiUTC_A[28];
PGSt_scTime scTime[8];
PGSt_SMF_status returnStatus;
/*
Begin example
*/

spacecraftID = EOS_AM;         /* or TRMM or EOS_PM */

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");

returnStatus = PGS_TD_UTC_to_SCTime( spacecraftID,
             asciiUTC_A, scTime );
/*
variable scTime now contains an 8-byte packed CCSDS
binary format value, corresponding to the UTC time
1998-06-30T10:51:28.320000Z
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_utc_to_sctime
      INTEGER spacecraftid
      CHARACTER*27 asciiutc_a
      CHARACTER*8 sctime
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      spacecraftid = EOS_AM        ! or TRMM or EOS_PM

      returnstatus = pgs_td_utc_to_sctime( spacecraftid,
     .                     asciiutc_a, sctime )

C variable sctime now contains an 8-byte packed CCSDS
C binary format value, corresponding to the UTC time
C 1998-06-30T10:51:28.320000Z
C
```

Notes:

Spacecraft clock time as output from this tool is the same format as returned by the Toolkit Level 0 Access tools (available in a future Toolkit delivery).

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.9 PGS_TD_UTCtoGPS

Short explanation of what it's for: Convert UTC time to GPS time.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoGPS.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double secGPS;
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoGPS( asciiUTC_A, &secGPS );
/*
variable secGPS now contains the value
583239101.320000
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_utctogps
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION secgps
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctogps( asciiutc_a, secgps )

C variable secgps now contains the value
C 583239101.320000
C
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.10 PGS_TD_UTCtoTAI

Short explanation of what it's for: Convert UTC time to TAI seconds from Jan 1, 1993.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoTAI.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double secTAI93;
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoTAI( asciiUTC_A, &secTAI93 );
/*
variable secTAI93 now contains the value
173357493.320000
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_utctotai
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION sectai93
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctotai( asciiutc_a, sectai93 )

C variable sectai93 now contains the value
C 173357493.320000
C
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.11 PGS_TD_UTCtoTDBjed

Short explanation of what it's for: Convert UTC time to Barycentric Dynamical time Julian date.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoTDBjed.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double jedTDB[2];
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoTDBjed( asciiUTC_A, jedTDB );
/*
variable jedTDB now contains the values
jedTDB[0] = 2450994.5        --  Half-integral TDB Julian date
jedTDB[1] = 0.45315398299    --  Fraction of TDB Julian date
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_utctotdbjed
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION jedtdb(2)
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctotdbjed( asciiutc_a, jedtdb )

C variable jedtdb now contains the values
C jedtdb(1) = 2450994.5        --  Half-integral TDB Julian date
C jedtdb(2) = 0.45315398299    --  Fraction of TDB Julian date
C
```

Notes:

Adding the two components of the output vector gives the full Julian date.

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.12 PGS_TD_UTCtoTDTjed

Short explanation of what it's for: Convert UTC time to Terrestrial Dynamical time Julian date.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoTDTjed.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double jedTDT[2];
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoTDTjed( asciiUTC_A, jedTDT );
/*
variable jedTDT now contains the values
jedTDT[0] = 2450994.5        --  Half-integral TDT Julian date
jedTDT[1] = 0.45315398148    --  Fraction of TDT Julian date
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INTEGER pgs_td_utctotdtjed
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION jedtdt(2)
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctotdtjed( asciiutc_a, jedtdt )

C variable jedtdt now contains the values
C jedtdt(1) = 2450994.5        --  Half-integral TDT Julian date
C jedtdt(2) = 0.45315398148    --  Fraction of TDT Julian date
C
```

Notes:

Adding the two components of the output vector gives the full Julian date.

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.13 PGS_TD_UTCtoUT1

Short explanation of what it's for: Convert Coordinated Universal Time (UTC) to Universal Time (UT1), in seconds since midnight.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoUT1.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double secUT1;
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoUT1( asciiUTC_A, &secUT1 );
/*
variable secUT1 now contains the value
39088.083809
seconds since midnight
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INTEGER pgs_td_utctout1
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION secut1
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctout1( asciiutc_a, secut1 )

C variable secut1 now contains the value
C 39088.083809
C seconds since midnight
C
```

Notes:

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

8.2.14 PGS_TD_UTCtoUT1jd

Short explanation of what it's for: Convert Coordinated Universal Time (UTC) to Universal Time (UT1), as a Julian Date.

This function is in file: $PGSSRC/TD/PGS_TD_UTCtoUT1jd.c

Examples:

Examples use CCSDS ASCII Time Code format A as the format of the input UTC. The function also accepts a time in CCSDS ASCII Time Code format B.

C example:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>
char asciiUTC_A[28];
PGSt_double jdUT1[2];
PGSt_SMF_status returnStatus;
/*
Begin example
*/

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );

returnStatus = PGS_TD_UTCtoUT1jd( asciiUTC_A, jdUT1 );
/*
variable jdUT1 now contains the values
jdUT1[0] = 2450994.5       --  Half-integral UT1 Julian date
jdUT1[1] = 0.452408392935  --  Fraction of UT1 Julian date
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INTEGER pgs_td_utctout1jd
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION jdut1(2)
      INTEGER returnstatus
C
C Begin example
C

      asciiutc_a = '1998-06-30T10:51:28.320000Z'

      returnstatus = pgs_td_utctout1jd( asciiutc_a, jdcut1 )

C
C variable jdut1 now contains the values
C jedtdb(1) = 2450994.5        --  Half-integral UT1 Julian date
C jedtdb(2) = 0.452408392935   --  Fraction of UT1 Julian date
C
```

Notes:

Adding the two components of the output vector gives the full UT1 Julian date.

See sec. 8.1.2, "Definition of Time Scales and Formats Used" for explanations of the time scales and formats.

9. Ancillary Data Access (AA) Tools

9.1 Ancillary Data Access (AA) Tools Overview

9.1.1 Introduction

The tools in this section are used to access ancillary data, i.e., data required for production processing which is obtained from independent external sources, and eventually other EOS products.

These tools are optional, in the sense that you may use your own functions to access this data if you so desire. The advantage to using the Toolkit functions is in reducing your coding effort, by providing geographic-type access to datasets having a geographic context. The tools are made to be as general as possible; however, you may still wish to write custom code, if the Toolkit functions do not entirely fill your needs.

The Toolkit functions divide into three groups: (1) tools which access vector format data, (2) tools which access gridded rectangular data, including digital elevation models (DEMs), and (3) tools which may be used by you to access an ASCII file. Group (1) consists only of a single tool that accesses a specific vector format file. Group (2) further divides into access of databases supplied with the Toolkit, and access to your own database using the Toolkit functions.

9.1.2 Accessing vector format data

There is one Toolkit function which accesses vector format data, PGS_AA_DCW. This function reads the land/sea/ice flag from the Digital Chart of the World (DCW) database only, a subset of which is supplied with the Toolkit delivery.

DCW is a general purpose digital global database designed for GIS (Geographical Information Systems) applications, with a scale of 1:1,000,000; it is in Vector Product Format (VPF). Essentially, for a given latitude and longitude, the Toolkit function will retrieve the land/sea/ice flag from the database corresponding to that position.

The DCW data files were installed in the directory of your choosing when you installed the Toolkit; this special handling is due to the fact that these files are large. These files are in subdirectories *eurnasia*, *noamer*, *soamafr*, and *sasaus*; their parent directory was specified by you when you installed the Toolkit.

In contrast to the gridded rectangular data access tools given below, function PGS_AA_DCW reads only DCW VPF data; it cannot be adapted to read an abritrary vector format file, unless you wish to go to the trouble of creating your own VPF database.

The DCW database contains other parameters, e.g., drainage and contour data; implementation of these awaits further requirements.

The reference documentation for DCW is *Digital Chart of the World -- Final DCW Product Specification MIL-D-89009, December 7, 1991*.
For information on VPF consult the document *Vector Product Format (MIL-STD-600006)*.
Both documents may be obtained from the Defense Mapping Agency Systems Center (AQE), 8613 Lee Highway, Fairfax, VA, 22031, Attn: Ms. Jean Rollins, Sr. Contract Specialist.

9.1.3 Accessing rectangular gridded data

These tools read datasets that are formatted as either 2- or 3-dimensional rectangular grids in various map projections. The map projections currently allowed are the equal angle (Platte Carre) projection and the NMC RUC model polar stereographic projection.

The tools include PGS_AA_DEM, PGS_AA_2DGEO and PGS_AA_3DGEO, which retrieve data for given latitudes and longitudes, and PGS_AA_2DRead and PGS_AA_3DRead, which access data for a given grid position coordinate or rectangular area. The tools may be used to either access the datasets supplied with the Toolkit, or alternatively for your own gridded rectangular datasets.

The Freeform software package from NOAA/NGDC has been adapted for Toolkit use in these tools. It is used internally by the Toolkit functions; the Freeform format is also used for formal data descriptions.

A note about FORTRAN: Because the internal C Toolkit gridded rectangular data access functions return the C *short* and *long* data types, for which the only corresponding ANSI FORTRAN 77 data type is *INTEGER*, there are separate files for the C and FORTRAN versions of these tools. However, the calling sequences are identical in C and FORTRAN.

9.1.3.1 Accessing the supplied rectangular gridded datasets

The datasets supplied with the Toolkit were selected on the basis of being the best currently available data, for which there were clear requirements from the various instrument teams. These datasets were supplied by NOAA's National Geophysical Data Center (NGDC).
In particular, the DEM datasets, namely DMA and TerrainBase, are the best available (as of Dec. 1994). The other datasets supplied with the Toolkit are old, and of low resolution; they are useful for prototyping and testing purposes, even if you do not plan to use them in your software at the DAAC. Additional datasets may be delivered in later versions of the Toolkit.

2-dimensional datasets

2D Rectangular Gridded Datasets Supplied With Toolkit

Data Set Units Cell size Filename **Olson World Ecosystems** v1.3a 30 cats 30 arc min owe13a.img v1.4d 74 cats 10 arc min owe14d.img v1.4dr 3 cats 10 arc min owe14dr.img v1.3a (Madagascar) 29 cats 30 arc min mowe13a.img **FNOC** modal elevation meters 10 arc min fnocmod.imgs* maximum elevation meters 10 arc min fnocmax.imgs* minimum elevation meters 10 arc min fnocmin.imgs* primary & secondary surface types 10 cats 10 arc min fnocpt.img ocean/land mask 2 cats 10 arc min fnococm.img number of ridges count 10 arc min fnocrdg.img direction of ridges degrees 10 arc min fnocazm.img water & urban cover percent 10 arc min fnocwat.img **Zobler** Soil types 108 cats 60 arc min srzsoil.img associated & included soil units 279 cats 60 arc min srzsubs.img* near surface soil texture 10 cats 60 arc min srztex.img surface slope 10 cats 60 arc min srzslop.img soil phase 87 cats 60 arc min srzphas.img special codes 12 cats 60 arc min srzcode.img world areas 9 cats 60 arc min srzarea.img **Etop05** surface elevation meters 5 arc min etop05.dat* **DMA** Conterminous USA meters 30 arc sec usatile# (# = 1 to 12) **Terrainbase global DEM** Complete meters 5 arc min tbase.bin Tiled meters 5 arc min tbase.xx (see notes)

Notes:

In the table, dataset file names marked with a "*" also have a separate file especially for use on the DEC workstation,; it is the same filename with "_dec" appended. This is due to the fact that on the DEC binary data is "byte-swapped", i.e., every two bytes are in reverse order than on all the other ECS-approved workstations.
(The DEMs do not need byte-swapped versions since the DEM tool does this internally.)

"cats" refers to the number of categories available in the dataset, in the returned value of the data.

In the dataset names listed in the table, FNOC stands for Fleet Numerical Ocenographic Center, Etop05 denotes Elevation Topographical 5 minutes, and DMA is Defense Mapping Agency.

The *Terrainbase global DEM, Tiled* database consists of four files: tbase.tl, tbase.tr, tbase.bl, tbase.br, corresponding to top left, top right, bottom left, and bottom right quadrants of a world map.

You may want to look at information about this data, obtained from the NOAA/NGDC WWW server.
General information about the CD-ROM is given in the Global Ecosystems Data on CD-ROM Flier SE-2006.
More specific info may be obtained in the Global Ecosystems Database, Version 1.0 (on CD-ROM) DISC-A. On the main menu, "Global (Geographic -- lat/long) Raster Data-Sets Description" section, the datasets supplied with the Toolkit are numbered

- A05 Olson World Ecosystems
- A11 Staub and Rosensweig Zobler Soil Type, Soil Texture, Surface Slope, and Other properties (Zobler)
- A13 FNOC Elevation, Terrain, and Surface Characteristics (FNOC)

(Information about the Etop05, DMA and TerrainBase datasets are given only on the CD-ROM itself.)

Hardcopy documentation consists of the *User's Guide* (EPA/600/R-92/194a) and the *Documentation Manual, Disc A* (EPA/600/R-92/194b) for the *Global Ecosystems Database, Version 1.0 (on CD-ROM)*, published by EPA Global Climate Research Program, NOAA/NGDC Global Change Database Program, NGDC Key to Geophysical Records Documentation No. 26, Incorporated in: Global Change Database, Vol. 1, NOAA/NGDC, 325 Broadway, Boulder, CO 80303, June 1992.

3-dimensional dataset

There is one additional dataset supplied, which is 3-dimensional. Named *nmcRucPotPres.datrepack* (*nmcRucPotPres.datrepack_dec* for DEC workstations), it is derived from a particular NMC Rapid Update Cycle (RUC) Analysis and Forecast System sequential dataset, forecast at 00Z, which is is GRIB format. The specific two parameters chosen for this test dataset are the potential temperature profile (POT, NMC parm #13, in deg.K) and the pressure profile (PRES, NMC parm #1, in Pa) at 4 sigma levels; they are in the NMC RUC model polar stereographic projection. (Sigma is the ratio of pressure to surface pressure.) The 4 sigma levels included are 1.0, 0.8, 0.6 and 0.4, in that order in the data file. These parameters were selected from a model run for a test period. **These data are intended for test purposes only, and are not generally applicable.**

For those interested, documentation of the GRIB format is available via anonymous ftp from the NOAA NMC public data server "nic" at *nic.fb4.noaa.gov* (140.90.50.22) in the directories */pub/nws/nmc/docs/gribguide* and */pub/nws/nmc/docs/gribed1*.
ECS' source for this information is the EOS document *Documentation for NOAA's NMC Gridded Data Products*, Version 0.1, 6 October 1994, by Matthew Schwaller *(matt@ulabsgi.gsfc.nasa.gov)*, Brian Krupp *(krupp@spso2.gsfc.nasa.gov)*, and Anand Swarroop.
Note that in general before use GRIB files must first be reformatted to simple binary using the available decoders.

Note: A discrepancy was discovered after the Toolkit package was delivered. The order of the parameters in format file *nmcRucSigPotPres.bfm* is incorrect. The file should look like this:

```
nmcRucSigPot 1 4 float 1
nmcRucSigPres 5 8 float 1
```

because this is the actual order of the data in the file. Please edit this file and reverse the order so that it looks like the above. Otherwise, the data returned will be reversed, e.g., the potential temperature will be in the pressure position in your output variable and vice versa.

Details of how to use the Toolkit to access the supplied datasets are given in the Tool Descriptions for the appropriate tools, including PGS_AA_2DGEO, PGS_AA_3DGEO, PGS_AA_2DRead, PGS_AA_3DRead and PGS_AA_DEM.
The first four of these access a single physical file; function PGS_AA_DEM may be used to access a group of physical files which are part of the same data set, i.e., in the same format.

For more details of how the Toolkit works internally to access these files, see the next section.

9.1.3.2 Accessing your own rectangular gridded datasets

You may use the Toolkit to access your own ancillary files, provided they are rectangular gridded datasets, in the equal angle (Platte Carre) projection or the NMC RUC model polar stereographic projection. In this section we explain how to do this step-by-step. We show how to prepare the format file, the support file, the Toolkit Process Control file, and the index file. (Definitions of these appear in the following sections.)

As an example, we show how the North America regional digital elevation model (DEM) from the TerrainBase CD-ROM would be prepared for access by Toolkit AA tools. TerrainBase is a new (8/94) worldwide digital terrain database from the U.S. Defense Mapping Agency, available from NOAA/NGDC.
The TerrainBase data set is now (3/95) part of the Toolkit delivery; so you wouldn't need to perform the actions given in this example to use that data.

A general overview of the Global View CD-ROM is available. Contact Allen M. Hittelman, Solid Earth Geophysics Division, NGDC, at *amh@ngdc.noaa.gov*.

The name of the North America file is *america.bin*. We choose to put this file in the default directory for PRODUCT INPUT files, viz. *$PGS_PRODUCT_INPUT*. (Please note that this file is not included in the Toolkit delivery. It is used here for illustrative purposes only.)
This example shows how to prepare a single physical file for Toolkit access.

This particular example file contains 2-byte data that is ordered with the low byte first. If you are using this particular file at the SCF, this means that unless you are on the DEC workstation, you have to translate the data file to put high byte first. A simple program that reads in every 2 bytes, then writes out the 2nd byte first and the 1st byte second, will do the trick. At the DAAC this function is done by preprocessing software, independent of your software and the Toolkit.

9.1.3.2.1 Preparing the Format File

First you need to make a **format file**. This text file contains information about the actual format of the main dataset. It is used by the Freeform software internally in the Toolkit. Each main data set has exactly one format file.

In our simple example, this file has only one line; it looks like:

```
americaSeaLevelElevM05 1 2 short 0
```

Explanation of parameters:

americaSeaLevelElevM05 -- Toolkit parameter ID stringString used as input to Toolkit functions (1st argument parms), when you want to retrieve this parameter. This string can be anything you like; here we identify the parameter as North America, 5 minute grid, sea level elevation, in meters.1 2 -- Input start and stop bytesStart and stop bytes of the parameter on the grid.short -- Input data type of the parameterSince this is used by Freeform, which is written in C, this is a C data type. Long, float and double are also valid. This value is machine dependent.0 -- reserved for future use

Only main datasets that are binary files are supported in this Toolkit release. The Freeform software requires such files to be named with suffix ".bfm", so you need to name this file accordingly. To be consistent with the main dataset name, we choose to call this file *america.bfm*. Also, we choose to put this file in directory $PGSHOME/runtime, where $PGSHOME is where you installed the Toolkit.

Interleaving of data is possible.

9.1.3.2.2 Preparing the Support File

Next you need to prepare a **support file**. This file contains metadata about the main dataset. A given support file may be used for many main datasets, as appropriate.

The text file which describes this subset of the TerrainBase data looks like this:

```
cacheFormat1        = short
cacheFormat2        = 0
cacheFormatBytes  = 2
parmMemoryCache        = 1362528
dataType        = short
autoOperation        = 0
fileMemoryCache        = 1362528
maxLat        = 65.0
minLat        = 5.0
maxLong        = -52.0
minLong        = -135.0
xCells        = 684
yCells        = 996
zCells        = 0
```

Explanation of parameters:

short -- cacheFormat1Data type you want as output from the Toolkit call. Long, float and double are also valid. In FORTRAN, if you use short or long in this field, the result is cast to PGSt_integer by the Toolkit.0 -- cacheFormat2reserved for future use2 -- cacheFormatBytesMachine-specific size of cacheFormat1 in output. If you do not know this, you might write a short program in C using the sizeof function to determine it; alternatively, many debuggers will supply this information.1362528 -- parmMemoryCacheSize in bytes of the output data for this parameter, which has type cacheFormat1.If there is only one parameter in the input file, and cacheFormat1 is the same as the input data type in the format file (see sec 9.1.3.2.1, "Preparing the Format File", for a description), then this value is the same as fileMemoryCache. Otherwise, you need to calculate the total number of bytes for this parameter.short -- dataTypeThis must be identical to cacheFormat1.1 -- autoOperationThis parameter is for applying operations to the main dataset, which are necessary to get the data out in the proper form. Operations currently available (with parameter value in parentheses) include calculating row cell coordinates from geographic coordinates assuming either equal angle (Platte Carre) (1) or NMC RUC model polar stereographic (2) projection, and recalculating geographic coordinates assuming longitude 0 at either Greenwich (4) or the International Date Line (8). More than one auto operation may be applied at once by summing the parameter values. See the "Auto operations" section in the Toolkit Users Guide, Appendix D, sec. 3.2.3 .For this particular data set, the equal angle (Platte Carre) function is applied to the raw data. 1362528 -- fileMemoryCacheSize in bytes of main dataset file america.bin65.0 -- maxLat5.0 -- minLat-52.0 -- maxLong-135.0 -- minLongMinimum and maximum latitude and longitude of main dataset684 -- xCells996 -- yCells0 -- zCellsNumber of cells in each dimension of main dataset.x direction is fastest changing, z direction is slowest

We choose to call this file *americaSupport*, and to put it in directory $PGSHOME/runtime.

The above description also covers all support files delivered with the Toolkit, with one exception: the NMC RUC 3D extracted test datasets. For these datasets, the following fields are added, using examples from the file *nmcRucSupport*:

22.8756 -- lowerLeftLatLower left latitude of the grid origin cell239.5089 -- lowerLeftLongLower left longitude of the grid origin cell, in E coordinates68153.0 -- meshLengthLength in meters of the cell255.0 -- gridOrientationGrid orientation in E coordinates

9.1.3.2.3 Preparing the Process Control File entries

This section explains the entries you need to make in the Toolkit Process Control file (PCF).

Before we start, please note that the logical identifer numbers that you use in the first column of the PCF must not be in the range 10000 - 10999, as these are reserved for internal Toolkit use. Also, in the PCF examples, the vertical ellipsis ". . ." refers to entries for other files.

First you need to put an entry for your **main data file** into the PCF. Here is what this entry would look like:

```
?   PRODUCT INPUT FILES
# [ set env var PGS_PRODUCT_INPUT for default location ]
.
.
.
501|america.bin|||||1
```

Here *501* is the logical file identifier you use as input to Toolkit functions in your code (via #define in C or PARAMETER in FORTRAN), america.bin is the name of your main data file, and *1* is the required version number. By default, this file resides in directory *$PGS_PRODUCT_INPUT*.

Next you need to put an entry in the PCF for your **support file**. This entry looks like

```
?   SUPPORT INPUT FILES
.
.
.
# ----------------------------------------------------------------
# These are support files for the data set files - to be created
# by user (not necessarily a one-to-one relationship)
# The IDs must correspond to the logical IDs in the index file
# -----------------------------------------------------------------
.
.
.
502|americaSupport|~/runtime||||1
```

The filename is $PGSHOME/runtime/americaSupport, where $PGSHOME is the directory where you installed the Toolkit.

You also need an entry in the PCF for your **format file**. This entry looks like

```
?   SUPPORT INPUT FILES
.
.
.
# --------------------------------------------------------------
# The following are format files for each data set file
# (not necessarily a one-to-one relationship)
# The IDs must correspond to the logical IDs in the index file
# --------------------------------------------------------------
.
.
.
503|america.bfm|~/runtime||||1
```

This file is named $PGSHOME/runtime/america.bfm.

For more details on using the PCF, see sec. 3.1.2, Constructing your Process Control file.

9.1.3.2.4 Preparing the Index File entry

You also need to make an entry in the **Index file**. This is an AA-tool specific file which maps the support and format files to the main dataset file. It already exists as *$PGSHOME/runtime/indexFile*, and contains entries for Toolkit-supplied gridded rectangular datasets.

First, edit the first line of this text file; add 1 to the number there. This is the number of entries in the file -- 23 are delivered with the Toolkit, so if you are adding one more, change it to 24.
Second, add an entry for your file to the end of the file. This looks like

```
americaSeaLevelElevM05                         502        503
```

Explanation of parameters:

americaSeaLevelElevM05 -- Toolkit parameter ID stringString used as input to Toolkit functions (1st argument parms), when you want to retrieve this parameter. Must be identical to the first field in the format file. (see sec 9.1.3.2.1, "Preparing the Format File", for a description.)502 -- Support file logical identifierMust be identical to the number used in field 1 of the PCF entry for the support file americaSupport. (see sec 9.1.3.2.3, "Preparing the Process Control File entries".)503 -- Format file logical identifierMust be identical to the number used in field 1 of the PCF entry for the format file america.bfm. (see sec 9.1.3.2.3, "Preparing the Process Control File entries".)

Now that you have done all this, you may use the Toolkit functions to retrieve the data. See the tool descriptions of PGS_AA_2DGEO, PGS_AA_3DGEO , PGS_AA_2DRead, PGS_AA_3DRead, and PGS_AA_DEM for further information.

9.1.4 Accessing data from an ASCII file

You may use a text file as an ancillary input file.

If this file has all its data in the format

```
PARAMETER=value
```

where *PARAMETER* is some keyword and *value* is its value, then you may use one of the PGS_AA_PeV* tools to read it. You pass it *PARAMETER* and it returns *value*.

Which function you use depends on what data type you want the returned value to be in:
Use *PGS_AA_PeV_string*, *PGS_AA_PeV_real*, or *PGS_AA_PeV_integer* to return the value as string, real or integer respectively.

9.2 Ancillary Data Access (AA) Tool Descriptions

9.2.1 PGS_AA_2DGEO

Short explanation of what it's for: Obtain data from a 2D rectangular gridded dataset for a given latitude and longitude.

This function is in file: $PGSSRC/AA/generic/PGS_AA_2DGEO.c and $PGSSRC/AA/generic/PGS_AA_2DGEOF.c

Examples:

Example uses the Etop05 dataset supplied with the Toolkit. Data is retrieved for 3 geographical locations.

C example:

```
#include <PGS_AA.h>

#define ETOP05 10955  /*Permanent logical ID for Etop05 dataset*/
#define NPARMS 1      /* No. parameters requested */
#define NPTS 3        /* No. points requested */
#define MAX_STRING 30 /* arbitrary */

char parms[NPARMS][MAX_STRING];
PGSt_double latitude[NPTS];
PGSt_double longitude[NPTS];
PGSt_integer version;
PGSt_integer operation;

short results[NPTS];  /* WARNING: This data type must be
                            identical to the cacheFormat1 field in
                            the support file */

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define parameter name desired
   WARNING: This string must be
      identical to the Toolkit parameter ID string field
      in both the format file
      and the index file entry */

strcpy( parms[0], "etop05SeaLevelElevM" );

latitude[0] = 51.5;
longitude[0] = 0.166666;

latitude[1] = 51.236666;
longitude[1] = 0.3832;

latitude[2] = 50.973333;
longitude[2] = 0.5999;

/* version corresponds to version number in the
   Process Control file entry for the main dataset
   Usual value is 1 */

version = 1;

/* Apply "nearest cell" operation; finds result at cell center
   (Currently this is the only allowed value for 2D datasets) */

operation = 1;

/* Call Toolkit function to find elevations at given lats/longs*/

returnStatus = PGS_AA_2DGEO( parms, NPARMS, latitude,
                  longitude, NPTS, ETOP05, version, operation,
                  results );

/*
Array results now contains the following elevations in meters:
results[0] = 20
results[1] = 64
results[2] =  1
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_2dgeo

      INTEGER ETOP05
      PARAMETER (ETOP05=10955) ! Permanent logical ID for Etop05
      INTEGER NPARMS
      PARAMETER (NPARMS=1)     ! No. parameters requested
      INTEGER NPTS
      PARAMETER (NPTS=3)       ! No. points requested

      CHARACTER*30 parms(NPARMS)

      DOUBLE PRECISION latitude(NPTS)
      DOUBLE PRECISION longitude(NPTS)
      INTEGER version
      INTEGER operation

C The data type of the results variable must correspond to the
C cacheFormat1 field in the support file as follows:
C
C cacheFormat1      results
C short            INTEGER
C long             INTEGER
C float            REAL
C double           DOUBLE PRECISION

      INTEGER results(NPTS)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter name desired
C WARNING: This string must be
C     identical to the Toolkit parameter ID string field in
C     in both the format file
C     and the index file entry

     parms(1) = 'etop05SeaLevelElevM'

      latitude(1) = 51.5
      longitude(1) = 0.166666

      latitude(2) = 51.236666
      longitude(2) = 0.3832

      latitude(3) = 50.973333
      longitude(3) = 0.5999

C version corresponds to version number in the
C  Process Control file entry for the main dataset
C  Usual value is 1

      version = 1

C Apply "nearest cell" operation; finds result at cell center
C (Currently this is the only allowed value for 2D datasets)

      operation = 1

C Call Toolkit function to find elevations at given lats/longs

      returnstatus = pgs_aa_2dgeo( parms, NPARMS, latitude,
     .               longitude, NPTS, ETOP05, version, operation,
     .               results )

C Array results now contains the following elevations in meters:
C results(1) = 20
C results(2) = 64
C results(3) =  1
```

Notes:

Currently, the input 2D dataset must be in the equal angle (Platte Carre) map projection in order for this tool to read it.

The number *NPTS* used for the dimension of the input and output variables must be **exactly** equal to the 5th argument in the calling sequence of the Toolkit function.

The next-to-last argument in the calling sequence *operation* is called the **user operation**; it specifies what additional functions you wish to apply to the data.
For this function, the value 1, which denotes operation PGS_AA_NEAREST_CELL, is currently the only available option.

Warning: Please make sure you have enough memory to access a given dataset. The Toolkit reads the entire dataset into memory at once. This may result in slow or erratic performance on machines with low memory available.
Note that the dataset *etop05.dat* of this example is 19 MB.
All other sample datasets supplied with the Toolkit are less than 5 MB.

The main dataset accessed in the example is *etop05.dat* (*etop05.dat_dec* if you have a DEC workstation). The format and support files for this dataset are *etop05.bfm* and *etop05Support* respectively; they are nominally located in directory $PGSHOME/runtime, unless you directed otherwise at Toolkit installation.
We reproduce the latter two files here for reference.
See sec. 9.1.3.2, " Accessing your own rectangular gridded datasets" for explanation of the parameters.

Listing of File *etop05.bfm*

etop05SeaLevelElevM 1 2 short 0

Listing of File *etop05Support*

cacheFormat1 = short cacheFormat2 = 0 cacheFormatBytes = 2 parmMemoryCache = 18662400 dataType = short autoOperation = 5 fileMemoryCache = 18662400 maxLat = 90.000 minLat = -90.0000 maxLong = 180.000 minLong = -180.000 xCells = 4320 yCells = 2160 zCells = 0

9.2.2 PGS_AA_2DRead

Short explanation of what it's for: Obtain data from a 2D rectangular gridded dataset for a given grid area.

This function is in file: $PGSSRC/AA/generic/PGS_AA_2DRead.c and $PGSSRC/AA/generic/PGS_AA_2DReadF.c

Examples:

Example uses the OlsonWorldEcosystems v1.3a dataset supplied with the Toolkit. Data is retrieved for a 2x3 cell grid.

C example:

```
#include <PGS_AA.h>

#define OWE13A 10952 /*Permanent logical ID for OWE v1.3a*/
#define NPARMS 1 /* No. parameters requested */
#define XDIM 2  /*Requested no.cells: faster changing direction*/
#define YDIM 3  /*Requested no.cells: slower changing direction*/
#define MAX_STRING 30 /* arbitrary */

char parms[NPARMS][MAX_STRING];
PGSt_integer xstart;
PGSt_integer ystart;
PGSt_integer version;
PGSt_integer operation;

short results[YDIM][XDIM];  /* WARNING: This data type must be
                            identical to the cacheFormat1 field in
                            the support file */

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define parameter name desired
   WARNING: This string must be
      identical to the Toolkit parameter ID string field
      in both the format file
      and the index file entry */

strcpy( parms[0], "OlsonWorldEcosystems1.3a" );

/* Define corner of grid requested */
xstart = 205;
ystart = 102;

/* version corresponds to the version number in the
   Process Control file entry for the main dataset
   Usual value is 1 */

version = 1;

/* This argument is reserved for future use */

operation = 0;

/* Call Toolkit function to find the category of the
      given grid area */

returnStatus = PGS_AA_2DRead( parms, NPARMS, xstart, ystart,
                  XDIM, YDIM, OWE13A, version, operation,
                  results );

/*
Matrix results now contains the values:
results[0][0] = 10
results[0][1] = 10
results[1][0] =  0
results[1][1] =  0
results[2][0] =  0
results[2][1] =  0

According to the Global Ecosystems Database documentation, the value
10 denotes category "Forest/Field; Dry Evergreen
broadleaf woods", while 0 denotes category "Oceans, Seas".

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_2dread

      INTEGER OWE13A
      PARAMETER (OWE13A=10952) ! Permanent logical ID - OWE v1.3a
      INTEGER NPARMS
      PARAMETER (NPARMS=1)     ! No. parameters requested
      INTEGER XDIM             ! Requested no. cells in
      PARAMETER (XDIM=2)       !    faster changing direction
      INTEGER YDIM             ! Requested no. cells in
      PARAMETER (YDIM=3)       !    slower changing direction

      CHARACTER*30 parms(NPARMS)
      INTEGER xstart
      INTEGER ystart
      INTEGER version
      INTEGER operation

C The data type of the results variable must correspond to the
C cacheFormat1 field in the support file as follows:
C
C cacheFormat1      results
C short           INTEGER
C long            INTEGER
C float           REAL
C double          DOUBLE PRECISION

      INTEGER results(XDIM)(YDIM)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter name desired
C WARNING: This string must be
C     identical to the Toolkit parameter ID string field in
C     in both the format file
C     and the index file entry

      parms(1) = 'OlsonWorldEcosystems1.3a'

C version corresponds to version number in the
C  Process Control file entry for the main dataset
C  Usual value is 1

      version = 1

C Reserved for future use

      operation = 0

C Call Toolkit function to find category of the
C     given grid area

      returnstatus = pgs_aa_2dread( parms, NPARMS, xstart, ystart,
     .                  XDIM, YDIM, OWE13A, version, operation,
     .                  results )

C Matrix results now contains the following elevations in meters:
C results(1)(1) = 10
C results(2)(1) = 10
C results(1)(2) =  0
C results(2)(2) =  0
C results(1)(3) =  0
C results(2)(3) =  0

C According to the Global Ecosystems Database documentation, the
C value 10 denotes category "Forest/Field; Dry Evergreen
C broadleaf woods", while 0 denotes category "Oceans, Seas".
```

Notes:

Currently, the input 2D dataset must be in the equal angle (Platte Carre) map projection in order for this tool to read it.

The numbers *XDIM* and *YDIM* used for the dimensions of the input and output variables must be **exactly** equal to the 5th and 6th arguments in the calling sequence of the Toolkit function.

The next-to-last argument in the calling sequence *operation* is called the **user operation**; it specifies what additional functions you wish to apply to the data.
For this function, this item is reserved for future use.

Warning: Please make sure you have enough memory to access a given dataset. The Toolkit reads the entire dataset into memory at once. This may result in slow or erratic performance on machines with low memory available.
Note that the dataset *owe13a.img* of this example is only 300 KB.
All other sample datasets supplied with the Toolkit are less than 5 MB, except *etop05.dat*, which is 19 MB.

The main dataset accessed in the example is *owe13a.img*. The format and support files for this dataset are *owe13a.bfm* and *owe13aSupport* respectively; they are nominally located in directory $PGSHOME/runtime, unless you directed otherwise at Toolkit installation.
We reproduce the latter two files here for reference.
See sec. 9.1.3.2, " Accessing your own rectangular gridded datasets" for explanation of the parameters.

Listing of File *owe13a.bfm*

OlsonWorldEcosystems1.3a 1 1 uchar 0

Listing of File *owe13aSupport*

cacheFormat1 = short cacheFormat2 = 0 cacheFormatBytes = 2 parmMemoryCache = 518400 dataType = short autoOperation = 9 fileMemoryCache = 259200 maxLat = 90.0000 minLat = -90.0000 maxLong = 180.000 minLong = -180.000 xCells = 720 yCells = 360 zCells = 0

9.2.3 PGS_AA_3DGEO

Short explanation of what it's for: Obtain data from a 3D rectangular gridded dataset for a given latitude and longitude.

This function is in file: $PGSSRC/AA/generic/PGS_AA_3DGEO.c and $PGSSRC/AA/generic/PGS_AA_3DGEOF.c

Examples:

Example uses the test data extracted from the NMC RUC dataset; this sample test dataset is supplied with the Toolkit.
Two parameters are extracted: Potential temperature profile and pressure profile at sigma level 1 (surface) in the test dataset.
Data are retrieved for 3 geographical locations.

C example:

```
#include <PGS_AA.h>

#define NMC_RUC_TEST 10972  /* Permanent logical ID for
                               NMC RUC test dataset */
#define NPARMS 2      /* No. parameters requested */
#define NPTS 3        /* No. points requested */
#define MAX_STRING 30 /* arbitrary */

char parms[NPARMS][MAX_STRING];
PGSt_double latitude[NPTS];
PGSt_double longitude[NPTS];
PGSt_integer sigma_level[NPTS];
PGSt_integer version;
PGSt_integer operation;

float results[NPTS][NPARMS];  /* WARNING: This data type must
                            be identical to the cacheFormat1 field
                            in the support file */

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define parameter names desired
   WARNING: This string must be
      identical to the Toolkit parameter ID string field
      in both the format file
      and the index file entry */

strcpy( parms[0], "nmcRucSigPot" ); /* Potential Temp profile */
strcpy( parms[1], "nmcRucSigPres" ); /* Pressure profile */

latitude[0] = 22.875610;
longitude[0] = -120.490300;
sigma_level[0] = 1;   /* corresponds to sigma=1.0 -- surface */

latitude[1] = 52.488790;
longitude[1] = -136.454700;
sigma_level[1] = 1;

latitude[2] = 46.017130;
longitude[2] = -60.828200;
sigma_level[2] = 1;

/* version corresponds to version number in the
   Process Control file entry for the main dataset
   Usual value is 1 */

version = 1;

/* Apply "nearest cell" operation for polar stereographic datasets
      Allows for uncertain boundary calculations
   Also available is "nearest cell" operation for equal area
      (Platte Carre) datasets (operation=1) */

operation = 2;

/* Call Toolkit function to find pressure and potential
      temperature at the given locations */

returnStatus =PGS_AA_3DGEO(parms, NPARMS, latitude, longitude,
          sigma_level, NPTS, NMC_RUC_TEST, version, operation,
          results );

/*
Matrix results now contains the values:
results[0][0] = 288.8      ! Potential Temperature in deg.K
results[0][1] = 101610.0   ! Pressure in Pa
results[1][0] = 275.7
results[1][1] = 103040.0
results[2][0] = 259.0
results[2][1] = 100560.0
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_3dgeo
```

```
      INTEGER NMC_RUC_TEST           ! Permanent logical ID for
      PARAMETER (NMC_RUC_TEST=10972) ! NMC RUC test dataset
      INTEGER NPARMS
      PARAMETER (NPARMS=2)     ! No. parameters requested
      INTEGER NPTS
      PARAMETER (NPTS=3)       ! No. points requested

      CHARACTER*30 parms(NPARMS)

      DOUBLE PRECISION latitude(NPTS)
      DOUBLE PRECISION longitude(NPTS)
      INTEGER sigma_level(NPTS)
      INTEGER version
      INTEGER operation

C The data type of the results variable must correspond to the
C cacheFormat1 field in the support file as follows:
C
C cacheFormat1      results
C short           INTEGER
C long            INTEGER
C float           REAL
C double          DOUBLE PRECISION

      REAL results(NPTS)(NPARMS)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter names desired
C WARNING: This string must be
C     identical to the Toolkit parameter ID string field in
C     in both the format file
C     and the index file entry

      parms(1) = 'nmcRucSigPot'   ! Potential Temperature profile
      parms(2) = 'nmcRucSigPres'  ! Pressure profile

      latitude(1) = 22.875610
      longitude(1) = -120.490300
      sigma_level(1) = 1  ! corresponds to sigma=1.0 -- surface

      latitude(2) = 52.488790
      longitude(2) = -136.454700
      sigma_level(2) = 1

      latitude(3) = 46.017130
      longitude(3) = -60.828200
      sigma_level(3) = 1

C version corresponds to version number in the
C  Process Control file entry for the main dataset
C  Usual value is 1

      version = 1

C Apply "nearest cell" operation for polar stereographic datasets
C     Allows for uncertain boundary calculations
C  Also available is "nearest cell" operation for equal area
C     (Platte Carre) datasets (operation=1) */

      operation = 2

C Call Toolkit function to find pressure and potential
C     temperature at the given locations

      returnstatus = pgs_aa_3dgeo( parms, NPARMS, latitude,
     .               longitude, sigma_level, NPTS, NMC_RUC_TEST, version, operation,
     .               results )

C Matrix results now contains the following values:
C results(1)(1) = 288.8    ! Potential Temperature in deg.K
C results(1)(2) = 101610.0 ! Pressure in Pa
C results(2)(1) = 275.7
C results(2)(2) = 103040.0
C results(3)(1) = 259.0
C results(3)(2) = 100560.0
```

Notes:

Currently, the input 3D dataset must be in either the NMC RUC polar stereographic or the equal angle (Platte Carre) map projection in order for this tool to read it.

The number *NPTS* used for the dimension of the input and output variables must be **exactly** equal to the 6th argument in the calling sequence of the Toolkit function.

Grid input variable *sigma_level* is the sigma level above the earth's surface, where sigma is the ratio of pressure to surface pressure. In the NMC RUC test dataset, 4 levels are provided: 1.0, 0.8, 0.6, and 0.4, in that order.

The next-to-last argument in the calling sequence *operation* is called the **user operation**; it specifies what additional functions you wish to apply to the data.
For this function, the value 1, which denotes operation PGS_AA_NEAREST_CELL, is available for equal angle (Platte Carre) datasets; the valuse 2, which denotes operation PGS_AA_OP_NINTCELL, is available for NMC RUC polar stereographic datasets.

Warning: Please make sure you have enough memory to access a given dataset. The Toolkit reads the entire dataset into memory at once. This may result in slow or erratic performance on machines with low memory available.
Note that the dataset *nmcRucPotPres.datrepack* of this example is only 160 KB.
All other sample datasets supplied with the Toolkit are less than 5 MB, except *etop05.dat*, which is 19 MB.

The main dataset accessed in the example is *nmcRucPotPres.datrepack*
(*nmcRucPotPres.datrepack_dec* if you have a DEC workstation). The format and support files for this dataset are *nmcRucSigPotPres.bfm* and *nmcRuc Support* respectively; they are nominally located in directory $PGSHOME/runtime, unless you directed otherwise at Toolkit installation. (Note that this support file applies to both potential temperature and pressure variables.)
We reproduce the latter two files here for reference.
See sec. 9.1.3.2, " Accessing your own rectangular gridded datasets" for explanation of the parameters.

Listing of File *nmcRucSigPotPres.bfm*

nmcRucSigPot 1 4 float 1 nmcRucSigPres 5 8 float 1

Note: A bug was discovered after the Toolkit package was delivered. The order of the parameters in the format file above were incorrectly reversed. Please edit this file and reverse the order so that it looks like the above. Otherwise, the data returned will be reversed, i.e., the potential temperature will be in the pressure position in your output variable and vice versa.

Listing of File *nmcRucSupport*

cacheFormat1 = float cacheFormat2 = 1 cacheFormatBytes = 4 parmMemoryCache = 80352 dataType = float autoOperation = 2 fileMemoryCache = 160704 maxLat = 0.000 minLat = 0.000 maxLong = 0.000 minLong = 0.000 xCells = 81 yCells = 62 zCells = 4 lowerLeftLat = 22.8756 lowerLeftLong = 239.5089 meshLength = 68153.0 gridOrientation = 255.0

9.2.4 PGS_AA_3DRead

Short explanation of what it's for: Obtain data from a 3D rectangular gridded dataset for a given grid area.

This function is in file: $PGSSRC/AA/generic/PGS_AA_3DRead.c and $PGSSRC/AA/generic/PGS_AA_3DReadF.c

Examples:

Example uses the test data extracted from the NMC RUC dataset; this sample test dataset is supplied with the Toolkit.
Two parameters are extracted: potential temperature profile and pressure profile.
Data are retrieved for a 2x3x2 cell block.

C example:

```
#include <PGS_AA.h>

#define NMC_RUC_TEST 10972  /* Permanent logical ID for
                               NMC RUC test dataset */
#define NPARMS 2 /* No. parameters requested */
#define XDIM 2 /*Requested no.cells: fastest changing direction*/
#define YDIM 3 /*Requested no.cells: middle changing direction*/
#define ZDIM 2 /*Requested no.cells: slowest changing direction*/
#define MAX_STRING 30 /* arbitrary */

char parms[NPARMS][MAX_STRING];
PGSt_integer xstart;
PGSt_integer ystart;
PGSt_integer zstart;
PGSt_integer version;
PGSt_integer operation;

/* WARNING: The data type of matrix results
       must be identical to the cacheFormat1 field in
       the support file */

float results[ZDIM][YDIM][XDIM][NPARMS];

PGSt_SMF_status returnStatus;
```

```
/* Begin example */

/* Define parameter names desired
   WARNING: This string must be
      identical to the Toolkit parameter ID string field
      in both the format file
      and the index file entry */

strcpy( parms[0], "nmcRucSigPot" ); /* Potential Temp profile */
strcpy( parms[1], "nmcRucSigPres" ); /* Pressure profile */

/* Define corner of grid block requested */
xstart = 30;
ystart = 20;
zstart =  2;

/* version corresponds to the version number in the
   Process Control file entry for the main dataset
   Usual value is 1 */

version = 1;

/* This argument is reserved for future use */

operation = 0;

/* Call Toolkit function to find the potential temperature
      and pressure profiles for the given grid block */

returnStatus = PGS_AA_3DRead( parms, NPARMS,
                  xstart, ystart, zstart, XDIM, YDIM, ZDIM,
                  NMC_RUC_TEST, version, operation,
                  results );

/*
Matrix results now contains the values:

(sigma level 1.0 -- surface)

results[0][0][0][0] = 302.6     ! Potential Temperature in deg.K
results[0][0][0][1] = 76830.0   ! Pressure in Pa
results[0][0][1][0] = 303.2
results[0][0][1][1] = 77390.0

results[0][1][0][0] = 302.3
results[0][1][0][1] = 76750.0
results[0][1][1][0] = 302.8
results[0][1][1][1] = 77580.0

results[0][2][0][0] = 303.2
results[0][2][0][1] = 75020.0
results[0][2][1][0] = 303.7
results[0][2][1][1] = 75540.0


(sigma level 0.8)

results[1][0][0][0] = 303.9
results[1][0][0][1] = 73830.0
results[1][0][1][0] = 304.4
results[1][0][1][1] = 74390.0

results[1][1][0][0] = 303.7
results[1][1][0][1] = 73750.0
results[1][1][1][0] = 304.3
results[1][1][1][1] = 74580.0

results[1][2][0][0] = 304.0
results[1][2][0][1] = 72020.0
results[1][2][1][0] = 304.6
results[1][2][1][1] = 72540.0

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_3dread

      INTEGER NMC_RUC_TEST           ! Permanent logical ID for
```

```
      PARAMETER (NMC_RUC_TEST=10972) ! NMC RUC test dataset
      INTEGER NPARMS
      PARAMETER (NPARMS=2)     ! No. parameters requested
      INTEGER XDIM             ! Requested no. cells in
      PARAMETER (XDIM=2)       !    fastest changing direction
      INTEGER YDIM             ! Requested no. cells in
      PARAMETER (YDIM=3)       !    middle changing direction
      INTEGER ZDIM             ! Requested no. cells in
      PARAMETER (ZDIM=2)       !    slowest changing direction

      CHARACTER*30 parms(NPARMS)
      INTEGER xstart
      INTEGER ystart
      INTEGER zstart
      INTEGER version
      INTEGER operation

C The data type of the results variable must correspond to the
C cacheFormat1 field in the support file as follows:
C
C cacheFormat1      results
C short           INTEGER
C long            INTEGER
C float           REAL
C double          DOUBLE PRECISION

      REAL results(XDIM)(YDIM)(ZDIM)(NPARMS)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter names desired
C WARNING: This string must be
C     identical to the Toolkit parameter ID string field in
C     in both the format file
C     and the index file entry

      parms(1) = 'nmcRucSigPot'  ! Potential Temperature profile
      parms(2) = 'nmcRucSigPres' ! Pressure profile

C version corresponds to version number in the
C  Process Control file entry for the main dataset
C  Usual value is 1

      version = 1

C Reserved for future use

      operation = 0

C Call Toolkit function to find the potential temperature
C     and pressure profiles for the given grid block


      returnstatus = pgs_aa_3dread( parms, NPARMS,
     .                  xstart, ystart, zstart, XDIM, YDIM, ZDIM,
     .                  NMC_RUC_TEST, version, operation,
     .                  results )

C Matrix results now contains the following values:


C (sigma level 1.0 -- surface)

C results(1)(1)(1)(1) = 302.6   ! Potential Temperature in deg.K
C results(1)(1)(1)(2) = 76830.0 ! Pressure in Pa
C results(1)(1)(2)(1) = 303.2
C results(1)(1)(2)(2) = 77390.0

C results(1)(2)(1)(1) = 302.3
C results(1)(2)(1)(2) = 76750.0
C results(1)(2)(2)(1) = 302.8
C results(1)(2)(2)(2) = 77580.0

C results(1)(3)(1)(1) = 303.2
C results(1)(3)(1)(2) = 75020.0
C results(1)(3)(2)(1) = 303.7
C results(1)(3)(2)(2) = 75540.0


C (sigma level 0.8)
```

```
C results(2)(1)(1)(1) = 303.9
C results(2)(1)(1)(2) = 73830.0
C results(2)(1)(2)(1) = 304.4
C results(2)(1)(2)(2) = 74390.0

C results(2)(2)(1)(1) = 303.7
C results(2)(2)(1)(2) = 73750.0
C results(2)(2)(2)(1) = 304.3
C results(2)(2)(2)(2) = 74580.0

C results(2)(3)(1)(1) = 304.0
C results(2)(3)(1)(2) = 72020.0
C results(2)(3)(2)(1) = 304.6
C results(2)(3)(2)(2) = 72540.0
```

Notes:

Currently, the input 3D dataset must be in either the NMC RUC polar stereographic or the equal angle (Platte Carre) map projection in order for this tool to read it.

The numbers *XDIM*, *YDIM* and *ZDIM* used for the dimensions of the input and output variables must be **exactly** equal to the 6th, 7th and 8th arguments in the calling sequence of the Toolkit function.

The next-to-last argument in the calling sequence *operation* is called the **user operation**; it specifies what additional functions you wish to apply to the data.
For this function, this item is reserved for future use.

Grid input variable *sigma_level* is the sigma level above the earth's surface, where sigma is the ratio of pressure to surface pressure. In the NMC RUC test dataset, 4 levels are provided: 1.0, 0.8, 0.6, and 0.4, in that order.

Warning: Please make sure you have enough memory to access a given dataset. The Toolkit reads the entire dataset into memory at once. This may result in slow or erratic performance on machines with low memory available.
Note that the dataset *nmcRucPotPres.datrepack* of this example is only 160 KB.
All other sample datasets supplied with the Toolkit are less than 5 MB, except *etop05.dat*, which is 19 MB.

The main dataset accessed in the example is *nmcRucPotPres.datrepack* (*nmcRucPotPres.datrepack_dec* if you have a DEC workstation). The format and support files for this dataset are *nmcRucSigPotPres.bfm* and *nmcRuc Support* respectively; they are nominally located in directory $PGSHOME/runtime, unless you directed otherwise at Toolkit installation. (Note that this support file applies to both potential temperature and pressure variables.)
We reproduce the latter two files here for reference.
See sec. 9.1.3.2, " Accessing your own rectangular gridded datasets" for explanation of the parameters.

Listing of File *nmcRucSigPotPres.bfm*

nmcRucSigPot 1 4 float 1 nmcRucSigPres 5 8 float 1

Note: A bug was discovered after the Toolkit package was delivered. The order of the parameters in the format file above were incorrectly reversed. Please edit this file and reverse the order so that it looks like the above. Otherwise, the data returned will be reversed, i.e., the potential temperature will be in the pressure position in your output variable and vice versa.

Listing of File *nmcRucSupport*

cacheFormat1 = float cacheFormat2 = 1 cacheFormatBytes = 4 parmMemoryCache = 80352 dataType = float autoOperation = 2 fileMemoryCache = 160704 maxLat = 0.000 minLat = 0.000 maxLong = 0.000 minLong = 0.000 xCells = 81 yCells = 62 zCells = 4 lowerLeftLat = 22.8756 lowerLeftLong = 239.5089 meshLength = 68153.0 gridOrientation = 255.0

9.2.5 PGS_AA_DEM

Short explanation of what it's for: Obtain data from a tiled 2D rectangular gridded dataset for a given latitude and longitude.
By 'tiled' is meant a dataset which is broken into more than one physical file. Primary use of this tool is for large digital elevation model (DEM) datasets.

This function is in file: $PGSSRC/AA/generic/PGS_AA_DEM.c and $PGSSRC/AA/generic/PGS_AA_DEMF.c

Examples:

Example uses the DMA Conterminous USA dataset supplied with the Toolkit. Data is retrieved for 3 geographical locations.

C example:

```
#include <PGS_AA.h>

#define DMA 10780     /*Permanent logical ID for DMA dataset*/
#define NPARMS 1      /* No. parameters requested */
#define NPTS 3        /* No. points requested */
#define MAX_STRING 30 /* arbitrary */

char parms[NPARMS][MAX_STRING];
PGSt_double latitude[NPTS];
PGSt_double longitude[NPTS];
PGSt_integer versionFlag[NPTS];
PGSt_integer operation;

short results[NPTS];  /* WARNING: This data type must be
                            identical to the cacheFormat1 field
                            in the support file */

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define parameter name desired
   WARNING: This string must be
      identical to the Toolkit parameter ID string field
      in both the format file
      and the index file entry;
      for tiled data sets, the numerical suffix is omitted
      in this string. */

strcpy( parms[0], "usadmaelevation" );

latitude[0] = 39.0;
longitude[0] = -77.0;

latitude[1] = 39.0;
longitude[1] = -106.0;

latitude[2] = 48.0;
longitude[2] = 0.0;

/* Apply "nearest cell" operation; finds result at cell center
   (Currently this is the only allowed value for 2D datasets) */

operation = 1;

/* Call Toolkit function to find elevations at given lats/longs*/

returnStatus = PGS_AA_DEM( parms, NPARMS, latitude,
                  longitude, versionFlag, NPTS, DMA, operation,
                  results );

/*
Array results now contains the following elevations in meters:
results[0] =   85
results[1] = 2829
results[2] =    0

Array versionFlag now contains the following values:
versionFlag[0] =  7  ! Point was found in file usatile7
versionFlag[1] =  6  ! Point was found in file usatile6
versionFlag[2] =  PGSd_AA_OUT_OF_RANGE  ! Point was not found in
                                          the DMA DEM data set

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_dem

      INTEGER DMA
      PARAMETER (DMA=10780) ! Permanent logical ID for DMA
      INTEGER NPARMS
      PARAMETER (NPARMS=1)     ! No. parameters requested
      INTEGER NPTS
      PARAMETER (NPTS=3)       ! No. points requested

      CHARACTER*30 parms(NPARMS)

      DOUBLE PRECISION latitude(NPTS)
      DOUBLE PRECISION longitude(NPTS)
      INTEGER versionflag(NPTS)
      INTEGER operation

C The data type of the results variable must correspond to the
C cacheFormat1 field in the support file as follows:
C
C cacheFormat1      results
C short            INTEGER
C long             INTEGER
C float            REAL
C double           DOUBLE PRECISION

      INTEGER results(NPTS)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter name desired
C WARNING: This string must be
C     identical to the Toolkit parameter ID string field in
C     in both the format file
C     and the index file entry;
C     for tiled data sets, the numerical suffix is omitted
C     in this string.

      parms(1) = 'usadmaelevation'

      latitude(1) = 39.0
      longitude(1) = -77.0

      latitude(2) = 39.0
      longitude(2) = -106.0

      latitude(3) = 48.0
      longitude(3) = 0.0

C Apply "nearest cell" operation; finds result at cell center
C (Currently this is the only allowed value for 2D datasets)

      operation = 1

C Call Toolkit function to find elevations at given lats/longs

      returnstatus = pgs_aa_dem( parms, NPARMS, latitude,
     .             longitude, versionflag, NPTS, DMA, operation,
     .             results )

C Array results now contains the following elevations in meters:
C results(1) =   85
C results(2) = 2829
C results(3) =    0

C Array versionFlag now contains the following values:
C versionFlag(1) =  7  ! Point was found in file usatile7
C versionFlag(2) =  6  ! Point was found in file usatile6
C versionFlag(3) =  PGSd_AA_OUT_OF_RANGE  ! Point was not found in
                                           the DMA DEM data set
```

Notes:

This function is essentially a wrapper on PGS_AA_2DGEO; the added functionality in this tool is that the data set may consist of more than one physical file.

This tool may be used to access data sets other than DEMs, if the user so desires.

Currently, the input 2D dataset must be in the equal angle (Platte Carre) map projection in order for this tool to read it.

The number *NPTS* used for the dimension of the input and output variables must be **exactly** equal to the 6th argument in the calling sequence of the Toolkit function.

The next-to-last argument in the calling sequence *operation* is called the **user operation**; it specifies what additional functions you wish to apply to the data.
For this function, the value 1, which denotes operation PGS_AA_NEAREST_CELL, is currently the only available option.

Warning: Please make sure you have enough memory to access a given dataset. The Toolkit reads the entire dataset into memory at once. This may result in slow or erratic performance on machines with low memory available.
All of the tiled DEM datasets supplied with the Toolkit are less than 5 MB.
The full (untiled) TerrainBase DEM, file *tbase.bin*, is 19 MB.

The main dataset accessed in the example is one of the family *usatile#*, where # = 1 to 12. The corresponding format and support files for this dataset are *usatile#.bfm* and *usatile#Support* respectively; they are nominally located in directory $PGSHOME/runtime, unless you directed otherwise at Toolkit installation.
For reference, we reproduce here an example of these files, viz. *usatile3.bfm* and *usatile3Support*.
See sec. 9.1.3.2, " Accessing your own rectangular gridded datasets" for explanation of the parameters.

`Listing of File` ***`usatile3.bfm`***

usadmaelevation3 1 2 short 0

`Listing of File` ***`usatile3Support`***

cacheFormat1 = short cacheFormat2 = 0 cacheFormatBytes = 2 parmMemoryCache = 4406400 dataType = short autoOperation = 9 fileMemoryCache = 4406400 maxLat = 51.000 minLat = 42.000 maxLong = -77.000 minLong = -94.000 xCells = 2040 yCells = 1080 zCells = 0

9.2.6 PGS_AA_DCW

Short explanation of what it's for: Obtain land/sea/ice flag from the Digital Chart of the World (DCW) dataset for a given latitude and longitude.

This function is in file: $PGSSRC/AA/DCW/PGS_AA_DCW.c

Examples:

Land/sea/ice flag is retrieved for 4 geographical locations.

C example:

```
#include <PGS_AA.h>

#define NPARMS 1   /* Currently only possible value */

char parms[NPARMS][MAX_STRING];
PGSt_double latitude[4];
PGSt_double longitude[4];
PGSt_integer npts;

PGSt_integer results[4];

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define parameter name desired; currently
      "po" ("political/oceans") is the only one allowed */

strcpy( parms[0], "po" );

latitude[0] = 70.0;
longitude[0] = 120.0;

latitude[1] = 50.0;
longitude[1] = -20.0;

latitude[2] = 85.55;
longitude[2] = 73.33;

latitude[3] = -69.22;
longitude[3] = 30.45;

npts = 4;

/* Call Toolkit function to find land/sea/ice flags
      at given lats/longs */

returnStatus = PGS_AA_DCW( parms, NPARMS, longitude,
                  latitude, npts, results );

/*
Array results now contains the following values:
results[0] = 1
results[1] = 2
results[2] = 3
results[3] = 4

Key to the returned values:

        Value          =        Surface Cover
        -----                -------------
       -1          =         No Data From DCW data base
        1          =        Land
        2          =        Open Ocean
        3          =        Polar Ice
        4          =         Pack Ice
        5          =        Shelf Ice
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_PC_9.f'

      INTEGER pgs_aa_dcw

      INTEGER NPARMS
      PARAMETER (NPARMS=1)     ! No. parameters requested

      CHARACTER*30 parms(NPARMS)

      DOUBLE PRECISION latitude(4)
      DOUBLE PRECISION longitude(4)
      INTEGER NPTS

      INTEGER results(4)

      INTEGER returnstatus
C
C Begin example
C

C Define parameter name desired; currently
C     'po' ("political/oceans") is the only one allowed

      parms(1) = 'po'

      latitude(1) = 70.0
      longitude(1) = 120.0

      latitude(2) = 50.0
      longitude(2) = -20.0

      latitude(3) = 85.55
      longitude(3) = 73.33

      latitude(4) = -69.22
      longitude(4) = 30.45

      npts = 4

C Call Toolkit function to find land/sea/ice flags
C     at given lats/longs

      returnstatus = pgs_aa_dcw( parms, NPARMS, longitude,
     .               latitude, npts, results )

C Array results now contains the following elevations in meters:
C results(1) = 1
C results(2) = 2
C results(3) = 3
C results(4) = 4

C Key to the returned values:

C        Value     =     Surface Cover
C        -----           -------------
C       -1         =     No Data From DCW data base
C        1         =     Land
C        2         =     Open Ocean
C        3         =     Polar Ice
C        4         =     Pack Ice
C        5         =     Shelf Ice
```

Notes:

It is much more efficient to call this tool for an array of input values, than to call it for one value at a time in a loop.

As this is an old (1991 or earlier) dataset, ice values are for illustrative purposes only.

See sec. 9.1.2, "Accessing vector format data" for more details about DCW.

9.2.7 PGS_AA_PeV*

Short explanation of what they're for: Obtain data from an ASCII (text) dataset in PARAMETER=value (PeV) format.

This description covers 3 tools: PGS_AA_PeV_string, PGS_AA_PeV_real, and PGS_AA_PeV_integer. These functions essentially do the same thing; they vary in the data type of the returned value.

These functions are in file: $PGSSRC/AA/generic/PGS_AA_PeV.c

Examples:

Example uses the OlsonWorldEcosystems v1.3a Support file dataset supplied with the Toolkit. This file actually contains metadata for the OlsonWorldEcosystems data; here we show how this tool retrieves data directly from it.
(This is done internally by Toolkit functions that access gridded rectangular data.)
A listing of this file appears in the Notes.

The intended use of these functions in science software is not to read metadata, but to read main ancillary datasets, as long as they are ASCII (text) files in PARAMETER=value format.

C example:

```
#include <PGS_AA.h>

#define OWE13A_SUPPORT 10902 /*ID for OWE v1.3a Support file*/
#define MAX_PARM_STRING 30 /* arbitrary */
#define MAX_VAL_STRING 100 /* arbitrary */

char parameter[MAX_PARM_STRING];

char value_s[MAX_VAL_STRING];
PGSt_double value_d;
PGSt_integer value_i;

PGSt_SMF_status returnStatus;

/* Begin example */

/******* Get a string value from the file *******/

/* Define parameter name desired */

strcpy( parameter, "cacheFormat1" );

/* Call Toolkit function to find the value of this
      parameter in the OWE v1.3a Support file */

returnStatus = PGS_AA_PeV_string( OWE13A_SUPPORT, parameter,
                  value_s );

/*
Variable value_s now contains the value "short"
*/

/******* Get a real value from the file *******/

/* Define parameter name desired */

strcpy( parameter, "maxLat" );

/* Call Toolkit function to find the value of this
      parameter in the OWE v1.3a Support file */

returnStatus = PGS_AA_PeV_real( OWE13A_SUPPORT, parameter,
                  &value_d );

/*
Variable value_d now contains the value 90.0
*/

/******* Get an integer value from the file *******/

/* Define parameter name desired */

strcpy( parameter, "parmMemoryCache" );

/* Call Toolkit function to find the value of this
      parameter in the OWE v1.3a Support file */

returnStatus = PGS_AA_PeV_integer( OWE13A_SUPPORT, parameter,
                  &value_i );

/*
 Variable value_i now contains the value 518400
*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_AA.f'
      INCLUDE 'PGS_AA_10.f'
      INCLUDE 'PGS_SMF.f'

      INTEGER pgs_aa_pev

      INTEGER OWE13A_SUPPORT
      PARAMETER (OWE13A_SUPPORT=10902) ! ID for OWE v1.3a
                                       ! Support file

      CHARACTER*30 parmameter

      CHARACTER*100 value_s
      DOUBLE PRECISION value_d
      INTEGER value_i

      INTEGER returnstatus
C
C Begin example
C

C******* Get a string value from the file *******

C Define parameter name desired

      parameter = 'cacheFormat1'

C Call Toolkit function to find the value of this
C     parameter in the OWE v1.3a Support file

      returnstatus = pgs_aa_pev_string( OWE13A_SUPPORT, parameter,
     .                  value_s )

C Variable value_s now contains the value 'short'


C******* Get a real value from the file *******

C Define parameter name desired

      parameter = 'maxLat'

C Call Toolkit function to find the value of this
C     parameter in the OWE v1.3a Support file

      returnstatus = pgs_aa_pev_real( OWE13A_SUPPORT, parameter,
     .            value_d )

C Variable value_d now contains the value 90.0


C******* Get an integer value from the file *******

     parameter = 'parmMemoryCache'

C Call Toolkit function to find the value of this
C      parameter in the OWE v1.3a Support file

      returnstatus = pgs_aa_pev_integer( OWE13A_SUPPORT, parameter,
     .           value_i )

C Variable value_i now contains the value 518400
```

Notes:

For a description of how the Process Control file is used (here by passing mnemonic *OWE13A_SUPPORT*), see sec 4.1.2, Constructing your Process Control file.

Listing of File *owe13aSupport*

cacheFormat1 = short cacheFormat2 = 0 cacheFormatBytes = 2 parmMemoryCache = 518400 dataType = short autoOperation = 9 fileMemoryCache = 259200 maxLat = 90.0000 minLat = -90.0000 maxLong = 180.000 minLong = -180.000 xCells = 720 yCells = 360 zCells = 0

10. Celestial Body Position (CBP) Tools

10.1 Celestial Body Position (CBP) Tools Overview

This section covers utilities you may use to obtain information about celestial bodies, including the Sun, Moon and planets. These tools are optional.

Tool PGS_CBP_body_inFOV determines whether the Sun, the Moon, a planet or a star is within the given field-of-view.

Tool PGS_CBP_Earth_CB_Vector determines the vector from the earth to a celestial body in the Earth-Centered Inertial (ECI) reference frame.

Tool PGS_CBP_Sat_CB_Vector calculates the vector in the spacecraft reference frame from the satellite to a celestial body.

Tool PGS_CBP_SolarTimeCoords computes various types of solar times, and also the Sun's position (right ascension and declination).
Solar times returned include:

Greenwich Mean Solar TimeTime based on the mean sun being overhead at noon at 0 deg. longitude (Greenwich). The fictitious mean Sun moves along the celestial equator at a constant rate of one revolution per year.Local Mean Solar TimeThis time is Greenwich Mean Solar Time adjusted for the longitude of observation, at one hour per 15 degrees.Local Apparent Solar TimeBased on the diurnal motion of the true Sun, whose rate varies seasonally due to the tilt of the earth's axis and the eccentricity of its orbit. Maximum annual difference with Local Mean Solar Time is 16 minutes. Adjusted for longitude of observation at one hour per 15 degrees.

All times are returned as seconds since midnight.

For brief descriptions of the ECI and spacecraft reference frames, see sec. 11.1, Coordinate Systems Conversion (CSC) Tools Overview.

All 4 tools use time in CCSDS ASCII Time Code format as one of their inputs. See section 8.1.2, "Definition of Time Scales and Formats Used", under the "UTC:Universal Coordinated Time" entry, for details of this format.

Information about the theoretical basis of these tools is available. If you are interested, please write to sdps-support@earthdata.nasa.gov.

10.2.2 PGS_CBP_Earth_CB_Vector

This section contains an alphabetical listing of the descriptions of the individual PGS_CBP_* tools.

10.2.1 PGS_CBP_body_inFOV

Short explanation of what it's for: Determine whether any part of a given celestial body is within the field-of-view (FOV), and return the S/C frame vector to that point.

This function is in file: $PGSSRC/CSC/PGS_CBP_body_inFOV.c

Examples:

It is determined whether the Moon is within the LIS instrument FOV, at a single time.

C example:

```
#include <PGS_CBP.h>

PGSt_integer nTimePts;
char asciiUTC_A[28];
PGSt_double time_offset[1];
PGSt_tag spacecraftID;
PGSt_integer numFovPerim;
PGSt_double fov_inside_vector[1][3];
PGSt_double fov_perim_vector[1][4][3];
PGSt_integer cb_id;
PGSt_double dummy[1][3];

PGSt_boolean bodyInFov_flag[1];
PGSt_double sc_body_vector[1][3];

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define base time and offsets desired.
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed.
   Offsets are in seconds.
   Offsets are useful if you want to determine whether a
   given point is in the FOV over some time interval.
   Here we process for a single time. */

nTimePts = 1;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;
```

```
/* Fill S/C frame vectors that define the field-of-view.
   Also supply a single arbitrary vector that is inside the FOV. */

numFovPerim = 4;

fov_perim_vector[0][0][0] = -0.534711; /* S/C frame X component */
fov_perim_vector[0][0][1] =  0.534711; /* S/C frame Y component */
fov_perim_vector[0][0][2] =  0.654345; /* S/C frame Z component */

fov_perim_vector[0][1][0] = -0.534711;
fov_perim_vector[0][1][1] = -0.534711;
fov_perim_vector[0][1][2] =  0.654345;

fov_perim_vector[0][2][0] =  0.534711;
fov_perim_vector[0][2][1] = -0.534711;
fov_perim_vector[0][2][2] =  0.654345;

fov_perim_vector[0][3][0] =  0.534711;
fov_perim_vector[0][3][1] =  0.534711;
fov_perim_vector[0][3][2] =  0.654345;

fov_inside_vector[0][0] = 0.0;
fov_inside_vector[0][1] = 0.0;
fov_inside_vector[0][2] = 0.654345;


/* Define the celestial body for which you want to determine
   whether it is in the FOV */

cb_id = PGSd_MOON;

/* Determine whether the Moon is in the FOV */

returnStatus = PGS_CBP_body_inFOV( nTimePts, asciiUTC_A,
                  time_offset, spacecraftID, numFovPerim,
                  fov_inside_vector, fov_perim_vector, cb_id,
                  bodyInFov_flag, dummy, sc_body_vector );

/* See the notes regarding "dummy".

   The following values are returned:


Flag that indicates if the celestial body is within the FOV

bodyInFov_flag[0] = PGS_FALSE


Vector from S/C to Moon in S/C frame coords (meters)

sc_body_vector[0,0] = -350156024.261   X coordinate
sc_body_vector[0,1] = -150805330.668   Y coordinate
sc_body_vector[0,2] =  130891208.962   Z coordinate

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_CBP.f'
      INCLUDE 'PGS_CBP_6.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_TD.f'

      INTEGER pgs_csc_body_infov

      INTEGER ntimepts
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(1)
      INTEGER spacecraftid
      INTEGER numfovperim
      DOUBLE PRECISION fov_inside_vector(3,1)
      DOUBLE PRECISION fov_perim_vector(3,4,1)
      INTEGER cb_id
      DOUBLE PRECISION dummy(3,1)

      INTEGER bodyinfov_flag(1)
      DOUBLE PRECISION sc_body_vector(3,1)

      INTEGER returnstatus
```

```
!
! Begin example
!
! Define base time and offsets desired.
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed.
!  Offsets are in seconds.
!  Offsets are useful if you want to determine whether a
!  given point is in the FOV over some time interval.
!  Here we process for a single time.

      ntimepts = 1
      asciiutc_a = '1998-06-30T10:51:28.320000Z'
      time_offset(1) = 0.0

! Assign spacecraft ID tag
!     PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
       spacecraftid = PGSd_TRMM


! Fill S/C frame vectors that define the field-of-view.
! Also supply a single arbitrary vector that is inside the FOV.

       fov_perim_vector(1,1,1) = -0.534711 ! S/C frame X pos
       fov_perim_vector(2,1,1) =  0.534711 ! S/C frame Y pos
       fov_perim_vector(3,1,1) =  0.654345 ! S/C frame Z pos

       fov_perim_vector(1,2,1) = -0.534711
       fov_perim_vector(2,2,1) = -0.534711
       fov_perim_vector(3,2,1) =  0.654345

       fov_perim_vector(1,3,1) =  0.534711
       fov_perim_vector(2,3,1) = -0.534711
       fov_perim_vector(3,3,1) =  0.654345

       fov_perim_vector(1,4,1) =  0.534711
       fov_perim_vector(2,4,1) =  0.534711
       fov_perim_vector(3,4,1) =  0.654345

       fov_inside_vector(1,1) = 0.0
       fov_inside_vector(2,1) = 0.0
       fov_inside_vector(3,1) = 0.654345

! Define the celestial body for which you want to determine
! whether it is in the FOV

       cb_id = PGSd_MOON

! Determine whether the first point is in the FOV

       returnstatus = pgs_csc_body_infov( ntimepts, asciiutc_a,
      +           time_offset, spacecraftid, numfovperim,
      +           fov_inside_vector, fov_perim_vector,
      +           bodyinfov_flag, dummy, sc_body_vector )

! See the notes regarding "dummy".

! The following values are returned:

! Flag that indicates if the celestial body is within the FOV

! bodyInFov_flag(1) = PGS_FALSE

! Vector from S/C to Moon in S/C frame coords (meters)

! sc_body_vector(1,1) = -350156024.261  X coordinate
! sc_body_vector(2,1) = -150805330.668  Y coordinate
! sc_body_vector(3,1) =  130891208.962  Z coordinate
```

Notes:

Below is a list of valid values for the 8th argument to this tool, the celestial body identifier *cb_id*. The mnemonic *PGSd_MOON* is used in the example; you may also use the numerical value (here 10), which may be useful to construct loops. These values are defined in file *$PGSINC/PGS_CBP.h* and *$PGSINC/PGS_CBP.f*. In the table, *SSBARY* denotes the solar system barycenter, and *EMBARY* denotes the barycenter of the Earth-Moon system. *STAR* denotes any point object.

```
      Celestial Body Identifiers

1 = PGSd_MERCURY           8 = PGSd_NEPTUNE
2 = PGSd_VENUS             9 = PGSd_PLUTO
3 = [unused]              10 = PGSd_MOON
4 = PGSd_MARS             11 = PGSd_SUN
5 = PGSd_JUPITER          12 = [unused]
6 = PGSd_SATURN           13 = [unused]
7 = PGSd_URANUS          999 = PGSd_STAR
```

The 10th argument of this tool *dummy* is ignored, unless the 8th argument *cb_id* is PGSd_STAR. In that case, the ECI vector to a point celestial body (e.g., a star) must be supplied in input variable *dummy*.

The finite sizes of all celestial objects (except PGSd_STAR) including satellites down to the 10th magnitude are taken into account by this function.

If errors in processing occur, the value PGSd_GEO_ERROR_VALUE is returned in the corresponding element of array *sc_body_vector*.

The number of points defining the FOV perimeter *numFovPerim* must be at least 3.
The perimeter vector points must be sequential around the FOV perimeter.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC
- JPL planetary ephemeris
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra, to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSLIB/database/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

10.2.2 PGS_CBP_Earth_CB_Vector

Short explanation of what it's for: Retrieve the ECI vector from the Earth to the Sun, the Moon or a planet for a given time or array of times.

This function is in file: $PGSSRC/CBP/PGS_CBP_Earth_CB_Vector.c

Examples:

ECI vector to Neptune is retrieved for 2 different times.

C example:

```
#include <PGS_CBP.h>

PGSt_integer npts;
char asciiUTC_B[26];
PGSt_double time_offset[2];
PGSt_integer cb_id;

PGSt_double eci_vector[2][3];

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code B format;
   CCSDS ASCII Time code A format is also allowed
   Offsets are in seconds */

strcpy( asciiUTC_B, "1994-012T13:46:21.45Z" );
time_offset[0] = 0.0;
time_offset[1] = 10.0;
npts = 2;

/* Define celestial body identifier for Neptune
   A list of possible values appears in the Notes */

cb_id = PGSd_NEPTUNE;

returnStatus = PGS_CBP_Earth_CB_Vector( npts, asciiUTC_B,
                       time_offset, cb_id, eci_vector );

/*
Matrix eci_vector now contains the following
ECI coordinates, in meters:

***For 1994-012T13:46:21.45Z***
eci_vector[0][0] = 1668891938932.883
eci_vector[0][1] = -4013391166628.138
eci_vector[0][2] = -1685932810433.269

***For 1994-012T13:46:31.45Z***
eci_vector[1][0] = 1668892270510.678
eci_vector[1][1] = -4013391044366.878
eci_vector[1][2] = -1685932759128.719

*/
```

FORTRAN example:

```
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CBP.f'
      INCLUDE 'PGS_CBP_6.f'
      INCLUDE 'PGS_TD_3.f'

      INTEGER pgs_cbp_earth_cb_vector

      INTEGER npts
      CHARACTER*25 asciiutc_b
      DOUBLE PRECISION time_offset(2)
      INTEGER cb_id

      DOUBLE PRECISION eci_vector(3,2)
      INTEGER returnstatus
!
! Begin example
!
! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code B format;
!  CCSDS ASCII Time code A format is also allowed
!  Offsets are in seconds
!
      asciiutc_b = '1994-012T13:46:21.45Z'
      time_offset(1) = 0.0
      time_offset(2) = 10.0
      npts = 2

! Define celestial body identifier for Neptune
!  A list of possible values appears in the Notes

       cb_id = PGSd_NEPTUNE

       returnStatus = pgs_cbp_earth_cb_vector( npts, asciiutc_b,
     +                  time_offset, cb_id, eci_vector )


! Matrix eci_vector now contains the following
! ECI coordinates, in meters:
!
! ***For 1994-012T13:46:21.45Z***
! eci_vector(1)(1) = 1668891938932.883
! eci_vector(2)(1) = -4013391166628.138
! eci_vector(3)(1) = -1685932810433.269
!
! ***For 1994-012T13:46:31.45Z***
! eci_vector(1)(2) = 1668892270510.678
! eci_vector(2)(2) = -4013391044366.878
! eci_vector(3)(2) = -1685932759128.719
!
```

Notes:

Below is a list of valid values for the 4th argument to this tool, the celestial body identifier *cb_id*. To use the names, prepend PGSd_ to the object, as in the present example. The mnemonic *PGSd_NEPTUNE* is used in the example; you may also use the numerical value (here 8), which may be useful to construct loops. These values are defined in file *$PGSINC/PGS_CBP.h* and *$PGSINC/PGS_CBP.f*. In the table, *PGSd_SSBARY* denotes the solar system barycenter, and *PGSd_EMBARY* denotes the barycenter of the Earth-Moon system.

```
               Celestial Body Identifiers

          1 = PGSd_MERCURY           8 = PGSd_NEPTUNE
          2 = PGSd_VENUS             9 = PGSd_PLUTO
          3 = [unused]              10 = PGSd_MOON
          4 = PGSd_MARS             11 = PGSd_SUN
          5 = PGSd_JUPITER          12 = PGSd_SSBARY
          6 = PGSd_SATURN           13 = PGSd_EMBARY
          7 = PGSd_URANUS
```

Files:

This tool accesses the following files:

- leap seconds
- JPL planetary ephemeris

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra, to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

10.2.3 PGS_CBP_Sat_CB_Vector

Short explanation of what it's for: Retrieve the spacecraft reference frame vector from the satellite to the Sun, the Moon or a planet for a given time.

This function is in file: $PGSSRC/CBP/PGS_CBP_Sat_CB_Vector.c

Examples:

Spacecraft reference frame vector to Neptune is retrieved for 2 different times.

C example:

```
#include <PGS_CBP.h>

PGSt_tag spacecraftID;
PGSt_integer npts;
char asciiUTC_B[26];
PGSt_double time_offset[2];
PGSt_integer cb_id;

PGSt_double sc_frame_vector[2][3];

PGSt_SMF_status returnStatus;

/* Begin example */

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code B format;
   CCSDS ASCII Time code A format is also allowed
   Offsets are in seconds */

strcpy( asciiUTC_B, "1994-012T13:46:21.45Z" );
time_offset[0] = 0.0;
time_offset[1] = 10.0;
npts = 2;

/* Define celestial body identifier for Neptune
   A list of possible values appears in the Notes */

cb_id = PGSd_NEPTUNE;

returnStatus = PGS_CBP_Sat_CB_Vector( spacecraftID, npts,
                    asciiUTC_B, time_offset, cb_id,
                    sc_frame_vector );

/*
Matrix sc_frame_vector now contains the following
spacecraft reference frame coordinates, in meters:

***For 1994-012T13:46:21.45Z***
sc_frame_vector[0][0] =   751956411224.327
sc_frame_vector[0][1] =  3871117053548.945
sc_frame_vector[0][2] = -2486772862890.086

***For 1994-012T13:46:31.45Z***
sc_frame_vector[1][0] =   723434403679.513
sc_frame_vector[1][1] =  3871139986897.981
sc_frame_vector[1][2] = -2495182523478.138

*/
```

FORTRAN example:

```
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CBP.f'
      INCLUDE 'PGS_CBP_6.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_TD.f'

      INTEGER pgs_cbp_sat_cb_vector

      INTEGER spacecraftid
      INTEGER npts
      CHARACTER*25 asciiutc_b
      DOUBLE PRECISION time_offset(2)
      INTEGER cb_id

      DOUBLE PRECISION sc_frame_vector(3,2)
      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code B format;
!  CCSDS ASCII Time code A format is also allowed
!  Offsets are in seconds
!
      asciiutc_b = '1994-012T13:46:21.45Z'
      time_offset(1) = 0.0
      time_offset(2) = 10.0
      npts = 2

! Define celestial body identifier for Neptune
!  A list of possible values appears in the Notes

       cb_id = PGSd_NEPTUNE

       returnStatus = pgs_cbp_sat_cb_vector( spacecraftid, npts,
     +                asciiutc_b, time_offset, cb_id,
     +                sc_frame_vector )


! Matrix sc_frame_vector now contains the following
! coordinates of NEPTUNE in the spacecraft reference frame,
! in meters:
!
! ***For 1994-012T13:46:21.45Z***
! sc_frame_vector(1)(1) =   751956411224.327
! sc_frame_vector(2)(1) =  3871117053548.945
! sc_frame_vector(3)(1) = -2486772862890.086
!
! ***For 1994-012T13:46:31.45Z***
! sc_frame_vector(1)(2) =   723434403679.513
! sc_frame_vector(2)(2) =  3871139986897.981
! sc_frame_vector(3)(2) = -2495182523478.138
!
```

Notes:

Below is a list of valid values for the 4th argument to this tool, the celestial body identifier *cb_id*. The mnemonic (*PGSd_NEPTUNE*) is used in the example; you may also use the numerical value (here 8), which may be useful to construct loops. These values are defined in file *$PGSINC /PGS_CBP.h*. In the table, *PGSd_SSBARY* denotes the solar system barycenter, and *PGSd_EMBARY* denotes the barycenter of the earth-moon system.

Celestial Body Identifiers

```
 1 = PGSd_MERCURY           8 = PGSd_NEPTUNE
 2 = PGSd_VENUS             9 = PGSd_PLUTO
 3 = PGSd_EARTH            10 = PGSd_MOON
 4 = PGSd_MARS             11 = PGSd_SUN
 5 = PGSd_JUPITER          12 = PGSd_SSBARY
 6 = PGSd_SATURN           13 = PGSd_EMBARY
 7 = PGSd_URANUS
```

Files:

This tool accesses the following files:

- leap seconds

- polar motion and UT1-UTC
- JPL planetary ephemeris
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra, to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSLIB/database/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

10.2.4 PGS_CBP_SolarTimeCoords

Short explanation of what it's for: Determine various types of solar time and also solar position for a given time.

This function is in file: $PGSSRC/CBP/PGS_CBP_SolarTimeCoords.c

Examples:

C example:

```
#include <PGS_CBP.h>

char asciiUTC_B[26];
PGSt_double longitude;

PGSt_double greenwich;
PGSt_double localMean;
PGSt_double localApparent;
PGSt_double rightAscension;
PGSt_double declination;

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define UTC time desired
   UTC time is given in CCSDS ASCII Time code B format;
   CCSDS ASCII Time code A format is also allowed */

strcpy( asciiUTC_B, "1994-012T13:46:21.45Z" );

/* Define longitude in radians -- positive is east of Greenwich
     Note: If you only want right ascension and declination,
     you may set this value to 0, since it is not used */

longitude = 1.0;

returnStatus = PGS_CBP_SolarTimeCoords( asciiUTC_B, longitude,
                       &greenwich, &localMean, &localApparent,
                       &rightAscension, &declination );

/*
The following values are returned:

Solar times in seconds since midnight:
greenwich=49506.859330      Greenwich Mean Solar Time
localMean=63257.846413      Local Mean Solar Time
localApparent=62758.715529  Local Apparent Solar Time

Solar coordinates:
rightAscension=5.098682     Right ascension of the Sun, radians
declination=-0.377383       Declination of the Sun, radians

*/
```

FORTRAN example:

```
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CBP.f'
      INCLUDE 'PGS_CBP_6.f'
      INCLUDE 'PGS_TD_3.f'

      INTEGER pgs_cbp_solartimecoords

      CHARACTER*25 asciiutc_b
      DOUBLE PRECISION longitude

      DOUBLE PRECISION greenwich
      DOUBLE PRECISION localmean
      DOUBLE PRECISION localapparent
      DOUBLE PRECISION rightascension
      DOUBLE PRECISION declination

      INTEGER returnstatus
!
! Begin example
!
! Define UTC time desired
!  UTC time is given in CCSDS ASCII Time code B format;
!  CCSDS ASCII Time code A format is also allowed */
!
      asciiutc_b = '1998-181T10:51:28.320000Z'

! Define longitude in radians -- positive is east of Greenwich
!    Note: If you only want right ascension and declination,
!    or the Greenwich solar time, you may set this value to 0.0,
!    since does not affect these answers.  Some number must be
!    supplied in any case because the function always calculates
!    all the results and could encounter data exceptions with
!    uninitialized input data

      longitude = 1.0

      returnStatus = pgs_cbp_solartimecoords( asciiUTC_B, longitude,
     .                 greenwich, localmean, localapparent,
     .                 rightascension, declination )

! The following values are returned:
!

! Solar times in seconds since midnight:
! greenwich=49506.859330      Greenwich Mean Solar Time
! localMean=63257.846413      Local Mean Solar Time
! localApparent=62758.715529  Local Apparent Solar Time

! Solar coordinates:
! rightAscension=5.098682     Right ascension of the Sun, radians
! declination=-0.377383       Declination of the Sun, radians
!
```

Notes:

Accuracy of the returned solar times is about 0.5 minutes; accuracy of the solar coordinates is about 0.7 milliradians (0.04 degrees).

See section 10.1, Celestial Body Position Tools Overview, for a description of the types of solar time returned by this tool.

11. Coordinate System Conversion (CSC) Tools

11.1 Coordinate System Conversion (CSC) Tools Overview

11.1.1 Introduction

These tools are used to manipulate various position and velocity coordinates in several reference frames. Use of them is optional.

The tools divide into two sections: tools that perform reference frame coordinate transformations directly, and tools that perform various other functions involving coordinate transformations.

Before discussing what the tools do, we give brief descriptions of the different reference frames used.

Reference Frames

Earth Centered Inertial (ECI)Inertial frame centered on earth, with Z axis along the rotational axis, and X axis directed toward vernal equinox. Epoch for all ECS data is J2000.Earth Centered Rotating (ECR)Frame with axes fixed in the earth, described by Cartesian coordinates. Z axis is along geographic North, and X axis is directed toward the Greenwich meridian.Geodetic (GEO)Frame with axes fixed in the earth, using geodetic latitude, longitude and altitude as coordinates.Orbital (ORB)Frame centered on the spacecraft, with the X-Z plane in the spacecraft orbital plane, and the Z axis directed toward geocentric nadir.Spacecraft (SC)Frame with axes fixed in the spacecraft. Coincides with Orbital frame when roll, pitch, and yaw are zero.

All toolkit functions use geodetic latitude for their latitude argument, if any. Geodetic latitude is defined as the angle a normal to the earth ellipsoid (earth model) makes with the equatorial plane, as opposed to the geocentric latitude, which is the angle that the vector to the center of the earth makes with the equatorial plane.

The earth models, or reference ellipsoids, used by some of these tools are defined in the "earth axis data file" *$PGSHOME/lib/database/CSC /earthfigure.dat*. You may edit this text file to add your own ellipsoid, if you like. The principal reference for the values supplied with the Toolkit is *Astronomical Almanac for the Year 1994,* U.S. Naval Observatory, 1993, p. K13, "Geodetic Reference Spheroids."
The book is available from the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402.

Many of the tools use time as an input. In the Toolkit, time is specified as UTC in CCSDS ASCII Time Code format, with optional offsets in seconds. See section 9.1.2, " Definition of Time Scales and Formats Used", under the "UTC: Universal Coordinated Time" entry, for details of this format.

11.1.2 Direct reference frame coordinate transformations

These tools perform coordinate transformations to and from the ECI, ECR, SC and ORB frames described above.

There are 10 of these. The function of each one is obvious from its name. Other transforms are possible using combinations of these. The transforms between Earth centered and spacecraft coordinate systems will produce only a rotation when the input vector is a unit vector, but will introduce the appropriate translation when the input is a vector in meters.

Tools that directly transform between reference frames

- PGS_CSC_ECItoECR
- PGS_CSC_ECItoORB
- PGS_CSC_ECItoSC
- PGS_CSC_ECRtoECI
- PGS_CSC_ECRtoGEO
- PGS_CSC_GEOtoECR
- PGS_CSC_ORBtoECI
- PGS_CSC_ORBtoSC
- PGS_CSC_SCtoECI
- PGS_CSC_SCtoORB

Note: In the May 1994 Toolkit delivery (TK2), some of these tools were combined in a single tool, *PGS_CSC_FrameChange*.

11.1.3 Other tools that involve coordinate systems

This section contains tools that perform various other functions involving coordinate transformations. It includes:

PGS_CSC_DayNightDetermines whether a given surface location at a given time is in day or night.PGS_CSC_Earthpt_FOVDetermines whether a given surface or atmosphere location at a given time is within a given field-of-view.PGS_CSC_GetFOV_PixelDetermines the footprint of an instrument field-of-view on the earth, and also returns some other related data.PGS_CSC_GreenwichHourFinds the hour angle of the vernal equinox at the Greenwich meridian.PGS_CSC_nutate2000Transforms a vector under nutation from Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Celestial Coordinates of date.PGS_CSC_precs2000Precesses a vector from Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Celestial Coordinates of date in Barycentric Dynamical Time (TDB).PGS_CSC_SpaceRefractEstimates the refraction for a ray incident from space or a line of sight from space to the Earth's surface, based on the unrefracted zenith angle.PGS_CSC_SubSatPointFinds the position and velocity of the sub-satellite point and rate of change of spacecraft altitude off the ellipsoid.PGS_CSC_wahr2Calculates nutation angles delta psi and delta epsilon, and their rates of change, referred to the ecliptic of date, from the Wahr series.PGS_CSC_ZenithAzimuthDetermines zenith angle and azimuth of instrument look vector or vector to a celestial body.

Information about the theoretical basis of these tools is available in **" Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project"**

11.2 Coordinate System Conversion (CSC) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_CSC_* tools.

11.2.1 PGS_CSC_DayNight

Short explanation of what it's for: Determine whether a given latitude/longitude at a given time is in day or night.

This function is in file: $PGSSRC/CBP/PGS_CSC_DayNight.c

Examples:

Whether two earth positions are in day or night is determined. The "NauticalNight" definition of day/night is used.

C example:

```
#include <PGS_CSC.h>

PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double latitude[2];
PGSt_double longitude[2];
PGSt_tag day_night_model;

PGSt_boolean is_dark[2];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Define time of the input lats/longs */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

latitude[0] = -0.603131     /* radians */
longitude[0] = 2.440543     /* radians */

latitude[1] = -0.603131
longitude[1] = 0.870543

/* Define day/night model
   See Notes for other possible values */

day_night_model = PGSd_NauticalNight;

/* Get day/night flags */

returnStatus = PGS_CSC_DayNight( numValues, asciiUTC_A,
                time_offset, latitude, longitude, day_night_model,
                is_dark );

/* Array is_dark now contains the following values:

is_dark[0] = PGS_TRUE;   This point is in the dark
is_dark[1] = PGS_FALSE;  This point is in the daylight

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CBP_6.f'
      INCLUDE 'PGS_CSC.f'
      INCLUDE 'PGS_CSC_4.f'

      INTEGER pgs_csc_daynight

      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION latitude(2)
      DOUBLE PRECISION longitude(2)
      INTEGER day_night_model

      INTEGER is_dark(2)

      INTEGER returnstatus
!
! Begin example
!

! Define time of the input lat/long

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0


      latitude(1) = -0.603131     ! radians
      longitude(1) = 2.440543     ! radians

      latitude(2) = -0.603131
      longitude(2) = 0.870543

! Define day/night model
! See Notes for other possible values

      day_night_model = PGSd_NauticalNight

! Get day/night flags

      returnstatus = pgs_csc_daynight( numvalues, asciiutc_a,
     .          time_offset, latitude, longitude, day_night_model,
     .          is_dark )


! Array is_dark now contains the following values:

! is_dark(1) = PGS_TRUE   This point is in the dark
! is_dark(2) = PGS_FALSE  This point is in the daylight
```

Notes:

Allowed values of the 6th argument in the calling sequence *day_night_model*:

PGSd_CivilTwilight(end of day) Sun deemed to set within 90 degrees 50 arc minutes from zenith.PGSd_CivilNight(end of civil twilight) Sun more than 96 degrees from zenith.(same as start of Nautical twilight)PGSd_NauticalNight(end of Nautical twilight) Sun more than 102 degrees from zenith. PGSd_AstronNight(end of Astronomical Twilight) Sun more than 108 degrees from zenith

A value other than PGS_TRUE or PGS_FALSE may be returned in the variable *is_dark* of the example. This indicates an error determining that value only; other elements of the output array are unaffected.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC
- earth model tags
- JPL planetary ephemeris

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.2 PGS_CSC_Earthpt_FOV

Short explanation of what it's for: Determine whether a given lat/long is within the field-of-view (FOV), and return the S/C frame vector to that point.

This function is in file: $PGSSRC/CBP/PGS_CSC_Earthpt_FOV.c

Examples:

It is determined whether a point is within the LIS instrument FOV, at a single time.

C example:

```
#include <PGS_CSC.h>

PGSt_integer nTimePts;
char asciiUTC_A[28];
PGSt_double time_offset[1];
PGSt_tag spacecraftID;
char earthModel[21];
PGSt_double latitude;
PGSt_double longitude;
PGSt_double altitude;
PGSt_double fov_inside_vector[1][3];
PGSt_integer numFovPerim;
PGSt_double fov_perim_vector[1][4][3];

PGSt_boolean ptInFov_flag[1];
PGSt_double sc_earthPt_vector[1][3];

PGSt_SMF_status returnStatus;

/* Begin example */

/* Define base time and offsets desired.
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed.
   Offsets are in seconds.
   Offsets are useful if you want to determine whether a
   given point is in the FOV over some time interval.
   Here we process for a single time. */

nTimePts = 1;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;

/* Assign spacecraft ID tag

    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define earth reference model */

strcpy( earthModel, "WGS84" );

/* Fill S/C frame vectors that define the field-of-view.
   Also supply a single arbitrary vector that is inside the FOV. */

numFovPerim = 4;

fov_perim_vector[0][0][0] = -0.534711; /* S/C frame X component */
fov_perim_vector[0][0][1] =  0.534711; /* S/C frame Y component */
fov_perim_vector[0][0][2] =  0.654345; /* S/C frame Z component */

fov_perim_vector[0][1][0] = -0.534711;
fov_perim_vector[0][1][1] = -0.534711;
fov_perim_vector[0][1][2] =  0.654345;

fov_perim_vector[0][2][0] =  0.534711;
fov_perim_vector[0][2][1] = -0.534711;
fov_perim_vector[0][2][2] =  0.654345;

fov_perim_vector[0][3][0] =  0.534711;
fov_perim_vector[0][3][1] =  0.534711;
fov_perim_vector[0][3][2] =  0.654345;

fov_inside_vector[0][0] = 0.0;
fov_inside_vector[0][1] = 0.0;
fov_inside_vector[0][2] = 0.654345;


/* Define the point for which you want to determine
   whether it is in the FOV */
```

```
latitude = -.64;
longitude = 2.46;
altitude = 0.0;

/* Determine whether the point is in the FOV */

returnStatus = PGS_CSC_Earthpt_FOV( nTimePts,
                 asciiUTC_A, time_offset, spacecraftID, earthModel,
                 latitude, longitude, altitude,
                 fov_inside_vector, numFovPerim, fov_perim_vector,
                 ptInFov_flag, sc_earthPt_vector );

/* The following values are returned:


Flag that indicates if given earth point is within the FOV

ptInFov_flag[0] = PGS_FALSE


Unit vector from S/C to earth point in S/C frame coords

sc_earthPt_vector[0,0] = -0.867   X coordinate
sc_earthPt_vector[0,1] =  0.031   Y coordinate
sc_earthPt_vector[0,2] =  0.497   Z coordinate

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'
      INCLUDE 'PGS_MEM_7.f'
      INCLUDE 'PGS_SMF.f'

      INTEGER pgs_csc_earthpt_fov

      INTEGER ntimepts
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(1)
      INTEGER spacecraftid
      CHARACTER*20 earthmodel
      DOUBLE PRECISION latitude
      DOUBLE PRECISION longitude
      DOUBLE PRECISION altitude
      DOUBLE PRECISION fov_inside_vector(3,1)
      INTEGER numfovperim
      DOUBLE PRECISION fov_perim_vector(3,4,1)

      INTEGER ptinfov_flag(1)
      DOUBLE PRECISION sc_earthpt_vector(3,1)

      INTEGER returnstatus

!
! Begin example
!
! Define base time and offsets desired.
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed.
!  Offsets are in seconds.
!  Offsets are useful if you want to determine whether a
!  given point is in the FOV over some time interval.
!  Here we process for a single time.

     ntimepts = 1
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0

! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define earth reference model

      earthModel = 'WGS84'


! Fill S/C frame vectors that define the field-of-view.
! Also supply a single arbitrary vector that is inside the FOV.
```

```
      fov_perim_vector(1,1,1) = -0.534711 ! S/C frame X pos
      fov_perim_vector(2,1,1) =  0.534711 ! S/C frame Y pos
      fov_perim_vector(3,1,1) =  0.654345 ! S/C frame Z pos

      fov_perim_vector(1,2,1) = -0.534711
      fov_perim_vector(2,2,1) = -0.534711
      fov_perim_vector(3,2,1) =  0.654345

      fov_perim_vector(1,3,1) =  0.534711
      fov_perim_vector(2,3,1) = -0.534711
      fov_perim_vector(3,3,1) =  0.654345

      fov_perim_vector(1,4,1) =  0.534711
      fov_perim_vector(2,4,1) =  0.534711
      fov_perim_vector(3,4,1) =  0.654345

      fov_inside_vector(1,1) = 0.0
      fov_inside_vector(2,1) = 0.0
      fov_inside_vector(3,1) = 0.654345

! Define the point for which you want to determine
! whether it is in the FOV

      latitude = -0.64
      longitude = 2.46
      altitude = 0.0

! Determine whether the point is in the FOV

      returnstatus = pgs_csc_earthpt_fov( ntimepts,
     +           asciiutc_a, time_offset, spacecraftid, earthmodel,
     +           latitude, longitude, altitude,
     +           fov_inside_vector, numfovperim, fov_perim_vector,
     +           ptinfov_flag, sc_earthpt_vector )

! The following values are returned:

! Flag that indicates if given earth point is within the FOV

! ptInFov_flag(1) = PGS_FALSE


! Unit vector from S/C to earth point in S/C frame coords

! sc_earthPt_vector(1,1) =  -0.867   X coordinate
! sc_earthPt_vector(2,1) =   0.031   Y coordinate
! sc_earthPt_vector(3,1) =   0.497   Z coordinate
```

Notes:

If errors in processing occur, the value PGSd_GEO_ERROR_VALUE is returned in the corresponding element of array *sc_earthPt_vector*.

The number of points defining the FOV perimeter *numFovPerim* must be at least 3.
The perimeter vector points must be sequential around the FOV perimeter.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC
- earth axis data
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.3 PGS_CSC_ECItoECR

Short explanation of what it's for: Convert a vector in Earth Centered Inertial (ECI) coordinates to Earth Centered Rotating (ECR) coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_ECItoECR.c

Examples:

Two ECI vectors containing position and velocity are converted to two ECR vectors.

C example:

```
#include <PGS_CSC.h>

PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double eci_vector[2][6];

PGSt_double ecr_vector[2][6];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 1.0;

/* Fill input vectors */

eci_vector[0][0] = -4191102.083176; /* ECI X pos, meters */
eci_vector[0][1] = -3647080.063050; /* ECI Y pos, meters */
eci_vector[0][2] = -3803463.200778; /* ECI Z pos, meters */
eci_vector[0][3] = 5402.13704;      /* ECI X vel, meters/sec */
eci_vector[0][4] = -5411.637312;    /* ECI Y vel, meters/sec */
eci_vector[0][5] = -764.857061;     /* ECI Z vel, meters/sec */

eci_vector[1][0] = -4185697.218627;
eci_vector[1][1] = -3652489.325640;
eci_vector[1][2] = -3804225.574655;
eci_vector[1][3] = 5407.590883;
eci_vector[1][4] = -5406.886691;
eci_vector[1][5] = -759.890523;

/* Get ECR vector */

returnStatus = PGS_CSC_ECItoECR( numValues, asciiUTC_A,
                       time_offset, eci_vector, ecr_vector );

/* Matrix ecr_vector now contains the following values:

ecr_vector[0][0] = -4245958.362002 ECR X pos, meters
ecr_vector[0][1] = 3583944.541294  ECR Y pos, meters
ecr_vector[0][2] = -3802636.071286 ECR Z pos, meters
ecr_vector[0][3] = -4259.539749    ECR X vel, meters/sec
ecr_vector[0][4] = -5857.188662    ECR Y vel, meters/sec
ecr_vector[0][5] = -765.490907     ECR Z vel, meters/sec

ecr_vector[1][0] = -4250215.575857
ecr_vector[1][1] = 3578085.340407
ecr_vector[1][2] = -3803399.079548
ecr_vector[1][3] = -4254.887147
ecr_vector[1][4] = -5861.211973
ecr_vector[1][5] = -760.525446

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'

      INTEGER pgs_csc_ecitoecr

      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION eci_vector(6,2)

      DOUBLE PRECISION ecr_vector(6,2)

      INTEGER returnstatus
!
! Begin example
!

!  Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 1.0

! Fill input vectors

      eci_vector(1,1) = -4191102.083176 ! ECI X pos, meters
      eci_vector(2,1) = -3647080.063050 ! ECI Y pos, meters
      eci_vector(3,1) = -3803463.200778 ! ECI Z pos, meters
      eci_vector(4,1) = 5402.137043     ! ECI X vel, meters/sec
      eci_vector(5,1) = -5411.637312    ! ECI Y vel, meters/sec
      eci_vector(6,1) = -764.857061     ! ECI Z vel, meters/sec

      eci_vector(1,2) = -4185697.218627
      eci_vector(2,2) = -3652489.325640
      eci_vector(3,2) = -3804225.574655
      eci_vector(4,2) = 5407.590883
      eci_vector(5,2) = -5406.886691
      eci_vector(6,2) = -759.890523

! Get ECR vector

      returnstatus = pgs_csc_ecitoecr( numvalues, asciiutc_a,
     .                 time_offset, eci_vector, ecr_vector )


! Matrix ecr_vector now contains the following values:

! ecr_vector(1,1) = -4245958.362002  ECR X pos, meters
! ecr_vector(2,1) = 3583944.541294   ECR Y pos, meters
! ecr_vector(3,1) = -3802636.071286  ECR Z pos, meters
! ecr_vector(4,1) = -4259.539749     ECR X vel, meters/sec
! ecr_vector(5,1) = -5857.188662     ECR Y vel, meters/sec
! ecr_vector(6,1) = -765.490907      ECR Z vel, meters/sec

! ecr_vector(1,2) = -4250215.575857
! ecr_vector(2,2) = 3578085.340407
! ecr_vector(3,2) = -3803399.079548
! ecr_vector(4,2) = -4254.887147
! ecr_vector(5,2) = -5861.211973
! ecr_vector(6,2) = -760.525446
```

Notes:

Epoch for the ECI input vector must be J2000.

Precession, nutation, and polar motion are all taken into account in the transformation.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.4 PGS_CSC_ECItoORB

Short explanation of what it's for: Convert a vector in Earth Centered Inertial (ECI) coordinates to Orbital (ORB) reference frame coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_ECItoORB.c

Examples:

Two ECI vectors containing position are converted to two ORB vectors.
The first is a spacecraft ephemeris ECI vector in meters; the second is a unit vector.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double eci_vector[2][3];

PGSt_double orb_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

eci_vector[0][0] =  1413531.574; /* ECI X pos, meters */
eci_vector[0][1] = -6005427.214; /* ECI Y pos, meters */
eci_vector[0][2] = -2693615.671; /* ECI Z pos, meters */

eci_vector[1][0] = -0.153457;    /* ECI unit vector */
eci_vector[1][1] =  0.482829;
eci_vector[1][2] =  0.862164;

/* Get ORB vector */

returnStatus = PGS_CSC_ECItoORB( spacecraftID, numValues,
                       asciiUTC_A, time_offset, eci_vector,
                       orb_vector );

/* Matrix orb_vector now contains the following values:

Since the first input vector was a spacecraft ephemeris
ECI vector, the ORB frame vector is the zero vector:


orb_vector[0][0] =  0.000  ORB X pos, meters
orb_vector[0][1] =  0.000  ORB Y pos, meters
orb_vector[0][2] =  0.000  ORB Z pos, meters

The second vector is a unit vector:

orb_vector[1][0] =  0.228986
orb_vector[1][1] = -0.545405
orb_vector[1][2] =  0.806287

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'

      INTEGER pgs_csc_ecitoorb

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION eci_vector(3,2)

      DOUBLE PRECISION orb_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      eci_vector(1,1) =  1413531.574  ! ECI X pos, meters
      eci_vector(2,1) = -6005427.214  ! ECI Y pos, meters
      eci_vector(3,1) = -2693615.671  ! ECI Z pos, meters

      eci_vector(1,2) = -0.153457     ! ECI unit vector
      eci_vector(2,2) =  0.482829
      eci_vector(3,2) =  0.862164

! Get ORB vector

      returnstatus = pgs_csc_ecitoorb( spacecraftid, numvalues,
     +                 asciiutc_a, time_offset, eci_vector,
     +                 orb_vector )


! Matrix orb_vector now contains the following values:

! Since the first input vector was a spacecraft ephemeris
! ECI vector, the ORB frame vector is the zero vector:

! orb_vector(1,1) =  0.000  ORB X pos, meters
! orb_vector(2,1) =  0.000  ORB Y pos, meters
! orb_vector(3,1) =  0.000  ORB Z pos, meters

! The second vector is a unit vector:

! orb_vector(1,2) =  0.228986
! orb_vector(2,2) = -0.545405
! orb_vector(3,2) =  0.806287
```

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.5 PGS_CSC_ECItoSC

Short explanation of what it's for: Convert a vector in Earth Centered Inertial (ECI) coordinates to Spacecraft (SC) reference frame coordinates .

This function is in file: $PGSSRC/CSC/PGS_CSC_ECItoSC.c

Examples:

Two ECI vectors containing position are converted to two SC vectors.
The first is a spacecraft ephemeris ECI vector in meters; the second is a unit vector.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double eci_vector[2][3];

PGSt_double sc_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

eci_vector[0][0] =  1413531.574; /* ECI X pos, meters */
eci_vector[0][1] = -6005427.214; /* ECI Y pos, meters */
eci_vector[0][2] = -2693615.671; /* ECI Z pos, meters */

eci_vector[1][0] = -0.153457;    /* ECI unit vector */
eci_vector[1][1] =  0.482829;
eci_vector[1][2] =  0.862164;

/* Get SC vector */

returnStatus = PGS_CSC_ECItoSC( spacecraftID, numValues,
                       asciiUTC_A, time_offset, eci_vector,
                       sc_vector );

/* Matrix sc_vector now contains the following values:

Since the first input vector was a spacecraft ephemeris
ECI vector, the S/C frame vector is the zero vector:

sc_vector[0][0] = 0.000  SC X pos, meters
sc_vector[0][1] = 0.000  SC Y pos, meters
sc_vector[0][2] = 0.000  SC Z pos, meters

The second vector is a unit vector:

sc_vector[1][0] =  0.228986
sc_vector[1][1] = -0.545405
sc_vector[1][2] =  0.806287

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'

      INTEGER pgs_csc_ecitosc

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION eci_vector(3,2)

      DOUBLE PRECISION sc_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      eci_vector(1,1) =  1413531.574 ! ECI X pos, meters
      eci_vector(2,1) = -6005427.214 ! ECI Y pos, meters
      eci_vector(3,1) = -2693615.671 ! ECI Z pos, meters

      eci_vector(1,2) = -0.153457       ! ECI unit vector
      eci_vector(2,2) =  0.482829
      eci_vector(3,2) =  0.862164

! Get SC vector

      returnstatus = pgs_csc_ecitosc( spacecraftid, numvalues,
     +                 asciiutc_a, time_offset, eci_vector,
     +                 sc_vector )


! Matrix sc_vector now contains the following values:

! Since the first input vector was a spacecraft ephemeris
! ECI vector, the S/C frame vector is the zero vector:

! sc_vector(1,1) = 0.000  SC X pos, meters
! sc_vector(2,1) = 0.000  SC Y pos, meters
! sc_vector(3,1) = 0.000  SC Z pos, meters

! The second vector is a unit vector:

! sc_vector(1,2) =  0.228986
! sc_vector(2,2) = -0.545405
! sc_vector(3,2) =  0.806287
```

Notes:

Aberration is taken into account in the transformation.

When the input vector is in meters, translation from earth center to spacecraft origin is accounted for. No translation is done when the input vector is a unit vector.

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.6 PGS_CSC_ECRtoECI

Short explanation of what it's for: Convert a vector in Earth Centered Rotating (ECR) coordinates to Earth Centered Inertial (ECI) coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_ECRtoECI.c

Examples:

Two ECR vectors containing position and velocity are converted to two ECI vectors.

C example:

```
#include <PGS_CSC.h>

PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double ecr_vector[2][6];

PGSt_double eci_vector[2][6];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 1.0;

/* Fill input vectors */

ecr_vector[0][0] = -4245958.362002; /*  ECR X pos, meters */
ecr_vector[0][1] = 3583944.541294;  /*  ECR Y pos, meters */
ecr_vector[0][2] = -3802636.071286; /*  ECR Z pos, meters */
ecr_vector[0][3] = -4259.539749;    /*  ECR X vel, meters/sec */
ecr_vector[0][4] = -5857.188662;    /*  ECR X vel, meters/sec */
ecr_vector[0][5] = -765.490907 ;    /*  ECR Z vel, meters/sec */

ecr_vector[1][0] = -4250215.575857
ecr_vector[1][1] = 3578085.340407
ecr_vector[1][2] = -3803399.079548
ecr_vector[1][3] = -4254.887147
ecr_vector[1][4] = -5861.211973
ecr_vector[1][5] = -760.525446


/* Get ECI vector */

returnStatus = PGS_CSC_ECRtoECI( numValues, asciiUTC_A,
                       time_offset, ecr_vector, eci_vector );

/* Matrix eci_vector now contains the following values:

eci_vector[0][0] = -4191102.083176   ECI X pos, meters
eci_vector[0][1] = -3647080.063050   ECI Y pos, meters
eci_vector[0][2] = -3803463.200778   ECI Z pos, meters
eci_vector[0][3] = 5402.13704        ECI X vel, meters/sec
eci_vector[0][4] = -5411.637312      ECI Y vel, meters/sec
eci_vector[0][5] = -764.857061       ECI Z vel, meters/sec

eci_vector[1][0] = -4185697.218627
eci_vector[1][1] = -3652489.325640
eci_vector[1][2] = -3804225.574655
eci_vector[1][3] = 5407.590883
eci_vector[1][4] = -5406.886691
eci_vector[1][5] = -759.890523

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'

      INTEGER pgs_csc_ecrtoeci

      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION ecr_vector(6,2)

      DOUBLE PRECISION eci_vector(6,2)

      INTEGER returnstatus
!
! Begin example
!

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 1.0

! Fill input vectors

      ecr_vector(1,1) = -4245958.362002 ! ECR X pos, meters
      ecr_vector(2,1) = 3583944.541294  ! ECR Y pos, meters
      ecr_vector(3,1) = -3802636.071286 ! ECR Z pos, meters
      ecr_vector(4,1) = -4259.539749    ! ECR X vel, meters/sec
      ecr_vector(5,1) = -5857.188662    ! ECR Y vel, meters/sec
      ecr_vector(6,1) = -765.490907     ! ECR Z vel, meters/sec

      ecr_vector(1,2) = -4250215.575857
      ecr_vector(2,2) = 3578085.340407
      ecr_vector(3,2) = -3803399.079548
      ecr_vector(4,2) = -4254.887147
      ecr_vector(5,2) = -5861.211973
      ecr_vector(6,2) = -760.525446

! Get ECI vector

      returnstatus = pgs_csc_ecrtoeci( numvalues, asciiutc_a,
     +                 time_offset, ecr_vector, eci_vector )

! Matrix eci_vector now contains the following values:

! eci_vector(1,1) = -4191102.083176   ECI X pos, meters
! eci_vector(2,1) = -3647080.063050   ECI Y pos, meters
! eci_vector(3,1) = -3803463.200778   ECI Z pos, meters
! eci_vector(4,1) = 5402.137043       ECI X vel, meters/sec
! eci_vector(5,1) = -5411.637312      ECI Y vel, meters/sec
! eci_vector(6,1) = -764.857061       ECI Z vel, meters/sec

! eci_vector(1,2) = -4185697.218627
! eci_vector(2,2) = -3652489.325640
! eci_vector(3,2) = -3804225.574655
! eci_vector(4,2) = 5407.590883
! eci_vector(5,2) = -5406.886691
! eci_vector(6,2) = -759.890523
```

Notes:

Epoch for the ECI output vector is J2000.

Precession, nutation, and polar motion are all taken into account in the transformation.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.7 PGS_CSC_ECRtoGEO

Short explanation of what it's for: Convert a vector in Earth Centered Rotating (ECR) coordinates to Geodetic (GEO) coordinates: latitude, longitude, and altitude.

This function is in file: $PGSSRC/CBP/PGS_CSC_ECRtoGEO.c

Examples:

A single ECR position vector is converted to geodetic coordinates.

C example:

```
#include <PGS_CSC.h>

PGSt_double ecr_vector[3];
char earthModel[21];

PGSt_double longitude;
PGSt_double latitude;
PGSt_double altitude;

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Fill input vector */

ecr_vector[0] = -4245958.362002; /* ECR X pos, meters */
ecr_vector[1] = 3583944.541294;  /* ECR Y pos, meters */
ecr_vector[2] = -3802636.071286; /* ECR Z pos, meters */

/* Define earth reference model */

strcpy( earthModel, "WGS84" );

/* Get long, lat, alt */

returnStatus = PGS_CSC_ECRtoGEO( ecr_vector, earthModel,
                       &longitude, &latitude, &altitude );


/* The following values are returned:

longitude = 2.440543       radians
latitude = -0.603131       radians
altitude = 361674.209546   meters

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'

      INTEGER pgs_csc_ecrtogeo

      DOUBLE PRECISION ecr_vector(3)
      CHARACTER*20 earthmodel

      DOUBLE PRECISION longitude
      DOUBLE PRECISION latitude
      DOUBLE PRECISION altitude

      INTEGER returnstatus
!
! Begin example
!

! Fill input vector

      ecr_vector(1) = -4245958.362002 ! ECR X pos, meters
      ecr_vector(2) = 3583944.541294  ! ECR Y pos, meters
      ecr_vector(3) = -3802636.071286 ! ECR Z pos, meters

! Define earth reference model

      earthmodel = 'WGS84'

! Get long, lat, alt

      returnstatus = pgs_csc_ecrtogeo( ecr_vector, earthmodel,
     +                 longitude, latitude, altitude )


! The following values are returned:

! longitude = 2.440543       radians
! latitude = -0.603131       radians
! altitude = 361674.209546   meters
```

Notes:

Input ECR vector must be in meters; it may not be a unit vector.

Files:

This tool accesses the following file:

- earth axis data

The physical reference to this file is defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to this file.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.8 PGS_CSC_GEOtoECR

Short explanation of what it's for: Convert a vector in Geodetic (GEO) coordinates (latitude, longitude, and altitude) to Earth Centered Rotating (ECR) coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_GEOtoECR.c

Examples:

A single set of geodetic latitude, longitude, and altitude is converted to an ECR position vector.

C example:

```
#include <PGS_CSC.h>

PGSt_double longitude;
PGSt_double latitude;
PGSt_double altitude;
char earthModel[21];

PGSt_double ecr_vector[3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Define input values */

longitude = 2.440543     /* radians */
latitude = -0.603131     /* radians */
altitude = 361674.209546 /* meters */

/* Define earth reference model */

strcpy( earthModel, "WGS84" );

/* Get ECR coordinates */

returnStatus = PGS_CSC_GEOtoECR( longitude, latitude,
                         altitude, earthmodel, ecr_vector );


/* Vector ecr_vector now contains the following values:

ecr_vector[0] = -4245955.860673   ECR X pos, meters
ecr_vector[1] =  3583944.676007   ECR Y pos, meters
ecr_vector[2] = -3802638.720370   ECR Z pos, meters

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'

      INTEGER pgs_csc_geotoecr

      DOUBLE PRECISION longitude
      DOUBLE PRECISION latitude
      DOUBLE PRECISION altitude
      CHARACTER*20 earthmodel

      DOUBLE PRECISION ecr_vector(3)

      INTEGER returnstatus
!
! Begin example
!

! Define input values

      longitude = 2.440543     ! radians
      latitude = -0.603131     ! radians
      altitude = 361674.209546 ! meters

! Define earth reference model
      earthmodel = 'WGS84'

! Get ECR coordinates

      returnstatus = pgs_csc_geotoecr( longitude, latitude,
     +                 altitude, earthmodel, ecr_vector )


! Vector ecr_vector now contains the following values:

!      ecr_vector(1) = -4245955.860673   ECR X pos, meters
!      ecr_vector(2) =  3583944.676007   ECR Y pos, meters
!      ecr_vector(3) = -3802638.720370   ECR Z pos, meters
```

Files:

This tool accesses the following file:

- earth axis data

The physical reference to this file is defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to this file.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.9 PGS_CSC_GetFOV_Pixel

Short explanation of what it's for: Determine where a field-of-view (FOV) projects on the earth's surface. Also return the pixel vectors in ECR for further use in the tool PGS_CSC_ZenithAzimuth() if desired. The field-of-view is specified by spacecraft frame pixel vectors.

This function is in file: $PGSSRC/CBP/PGS_CSC_GetFOV_Pixel.c

Examples:

Data about the earth projection of a square field of view is computed; taken from the LIS instrument.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[4];
char earthModel[21];
PGSt_boolean accuracy_flag;
PGSt_double sc_look_vector[4][3];
PGSt_double sc_offset[4][3];

PGSt_double latitude[4];
PGSt_double longitude[4];
PGSt_double ecr_unit_vector[4][3];
PGSt_double range[4];
PGSt_double range_rate[4];

PGSt_SMF_status returnStatus;

PGSt_integer i,j;

/*
Begin example
*/

/* Assign spacecraft ID tag
    PGS_dEOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired.
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed.
   Offsets in seconds are all set to 0.0 as we want a
   snapshot of the FOV. In general, when many times are to
   be processed, for staring instruments, the individual
   pixel vectors will still be fixed in the spacecraft
   frame, while for slewing instruments they must accurately
   reflect the scan */

numValues = 4;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
for (i=0;i<numValues;i++) time_offset[i] = 0.0;

/* Define earth reference model */

strcpy( earthModel, "WGS84" );

/* Set accuracy flag:

   Use PGS_FALSE for faster computation if you
   don't care either about Earth rotation during the light
   travel time or your instrument's offset from Spacecraft
   center.

   Use PGS_TRUE if you want to account for the
   earth's rotation during the light travel time. */

accuracy_flag = PGS_TRUE;

/* Fill S/C frame vectors that define the field-of-view.
   You could also use this to refer to individual pixels of
   your instrument. */
```

```
sc_look_vector[0][0] = -0.534711; /* S/C frame X component */
sc_look_vector[0][1] =  0.534711; /* S/C frame Y component */
sc_look_vector[0][2] =  0.654345; /* S/C frame Z component */

sc_look_vector[1][0] = -0.534711;
sc_look_vector[1][1] = -0.534711;
sc_look_vector[1][2] =  0.654345;

sc_look_vector[2][0] =  0.534711;
sc_look_vector[2][1] = -0.534711;
sc_look_vector[2][2] =  0.654345;

sc_look_vector[3][0] =  0.534711;
sc_look_vector[3][1] =  0.534711 ;
sc_look_vector[3][2] =  0.654345;

/* Set the pixel offsets.  These are for high accuracy; they locate
   the origin of a pixel with respect to the center-of-mass of the
   spacecraft.  Here we assume the instrument boresight is 15 meters
   off nominal center in the -y (orbit normal) direction. This array
   is used only if accuracy_flag is equal to PGS_TRUE.
   Naturally, only the part of the offset perpendicular to the
   boresight direction itself matters. */

for (i=0;i<4;i++)
{
     sc_offset[i][0] =   0.0;
     sc_offset[i][1] = -15.0;
     sc_offset[i][2] =   0.0;
}

/*  When accuracy_flag = PGS_FALSE you can Zero out instrument
    offsets, as they are not used, or you can ignore them and (in C)
    pass in a NULL pointer for the boresight offsets. (In FORTRAN
    pass in anything - for example, 0.0 )*/

/* Get data about the FOV projection on the earth */

returnStatus = PGS_CSC_GetFOV_Pixel( spacecraftID, numValues,
                       asciiUTC_A, time_offset, earthModel,
                       accuracy_flag, sc_look_vector, sc_offset,
                       latitude, longitude, ecr_unit_vector,
                       range, range_rate );

/* The following values are returned:

Latitudes of the FOV projection on the earth (radians):
latitude[0] = -0.478822
latitude[1] = -0.392320
latitude[2] = -0.350126
latitude[3] = -0.434660

Longitudes of the FOV projection on the earth (radians):
longitude[0] = -2.780467
longitude[1] = -2.825456
longitude[2] = -2.734285
longitude[3] = -2.685652

ECR reference frame representation of the input FOV vectors
ecr_unit_vector[0][0] =  0.73373160233  ECR X component
ecr_unit_vector[0][1] =  0.55040569686  ECR Y component
ecr_unit_vector[0][2] = -0.39836102296  ECR Z component

ecr_unit_vector[1][0] =  0.20219599731
ecr_unit_vector[1][1] =  0.85458983269
ecr_unit_vector[1][2] =  0.47832310892

ecr_unit_vector[2][0] =  0.38065007636
ecr_unit_vector[2][1] = -0.10337164875
ecr_unit_vector[2][2] =  0.91892318591

ecr_unit_vector[3][0] =  0.91220027384
ecr_unit_vector[3][1] = -0.40756416996
ecr_unit_vector[3][2] =  0.04221501817

Distance from spacecraft to earth intersection pt, meters
range[0] = 570980.678
range[1] = 564761.571
range[2] = 565402.381
range[3] = 571415.889

Velocity of the intersection pt in the ECR frame,
   projected along the look vector direction, meters/sec
```

```
range_rate[0] =  3991.164
range_rate[1] =  3786.974
range_rate[2] = -4008.894
range_rate[3] = -3804.699

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_getfov_pixel

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(4)
      CHARACTER*20 earthmodel
      INTEGER accuracy_flag
      DOUBLE PRECISION sc_look_vector(3,4)
      DOUBLE PRECISION sc_offset(3,4)

      DOUBLE PRECISION latitude(4)
      DOUBLE PRECISION longitude(4)
      DOUBLE PRECISION ecr_unit_vector(3,4)
      DOUBLE PRECISION range(4)
      DOUBLE PRECISION range_rate(4)

      INTEGER returnstatus

      INTEGER i,j
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired.
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed.
!  Offsets are in seconds.

     numvalues = 4
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     do i=1,numvalues
        time_offset(i) = 0.0
     enddo

! Define earth reference model

      earthModel = 'WGS84'

! Set accuracy flag
!  Use PGS_TRUE if you want to account for the
!  earth's rotation during the
!  time it takes light to travel

      accuracy_flag = PGS_FALSE

! Fill S/C frame vectors that define the field-of-view.
!   You could also use this to refer to individual pixels of
!   your instrument.

      sc_look_vector(1,1) = -0.534711 ! S/C frame X pos
      sc_look_vector(2,1) =  0.534711 ! S/C frame Y pos
      sc_look_vector(3,1) =  0.654345 ! S/C frame Z pos

      sc_look_vector(1,2) = -0.534711
      sc_look_vector(2,2) = -0.534711
      sc_look_vector(3,2) =  0.654345

      sc_look_vector(1,3) =  0.534711
      sc_look_vector(2,3) = -0.534711
      sc_look_vector(3,3) =  0.654345

      sc_look_vector(1,4) =  0.534711
      sc_look_vector(2,4) =  0.534711
```

```
      sc_look_vector(3,4) =  0.654345

! If the accuracy flag is set to PGS_FALSE, zero out instrument
! offsets, as they are not used, and you can even leave out the
! storage allocation and pass in anything.

!  Set the pixel offsets.  The are for high accuracy; they locate
!   the origin of a pixel with respect to the center-of-mass of the
!   spacecraft.  Here we assume the instrument boresight is 15 meters
!   off nominal center in the -y (orbit normal) direction. This array
!   is used only if accuracy_flag is equal to PGS_TRUE.

      do j=1,4
         sc_offset(1,j) =   0.0
         sc_offset(2,j) = -15.0
         sc_offset(3,j) =   0.0
      enddo

! Get earth intersection point data

      returnstatus = pgs_csc_getfov_pixel(spacecraftid, numvalues,
     .                 asciiutc_a, time_offset, earthmodel,
     .                 latitude, longitude, ecr_unit_vector,
     .                 range, range_rate )

! The following values are returned:

! Latitudes of the FOV projection on the earth (radians):
! latitude(1) = -0.478822
! latitude(2) = -0.392320
! latitude(3) = -0.350126
! latitude(4) = -0.434660

! Longitudes of the FOV projection on the earth (radians):
! longitude(1) = -2.780467
! longitude(2) = -2.825456
! longitude(3) = -2.734285
! longitude(4) = -2.685652


! ECR reference frame representation of the input FOV vectors
! ecr_unit_vector(1,1) =  0.73373160233  ECR X component
! ecr_unit_vector(2,1) =  0.55040569686  ECR Y component
! ecr_unit_vector(3,1) = -0.39836102296  ECR Z component

! ecr_unit_vector(1,2) =  0.20219599731
! ecr_unit_vector(2,2) =  0.85458983269
! ecr_unit_vector(3,2) =  0.47832310892

! ecr_unit_vector(1,3) =  0.38065007636
! ecr_unit_vector(2,3) = -0.10337164875
! ecr_unit_vector(3,3) =  0.91892318591

! ecr_unit_vector(1,4) =  0.91220027384
! ecr_unit_vector(2,4) = -0.40756416996
! ecr_unit_vector(3,4) =  0.04221501817


! Distance from spacecraft to earth intersection pt, meters
! range(1) = 570980.678
! range(2) = 564761.571
! range(3) = 565402.381
! range(4) = 571415.889

! Velocity of the intersection pt in the ECR frame,
!    projected along the look vector direction, meters/sec
! range_rate(1) =  3991.164
! range_rate(2) =  3786.974
! range_rate(3) = -4008.894
! range_rate(4) = -3804.699
```

Notes:

For more information about the *accuracy_flag* argument, see the Notes section of the Toolkit Users Guide entry for this tool (sec. 6.3.3).

The output value *ecr_unit_vector*, the ECR frame representation of the input SC frame look vector, may be useful for several things, including use as input to the tool PGS_CSC_ZenithAzimuth .

The values *range* and *range_rate* returned by this function are measures of the same data measured by Doppler radar instruments.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC
- earth model tags
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.10 PGS_CSC_GreenwichHour

Short explanation of what it's for: Determine the hour angle of the vernal equinox at the Greenwich meridian.

This function is in file: $PGSSRC/CBP/PGS_CSC_GreenwichHour.c

Examples:

Two Greenwich hour angles are determined.

C example:

```
#include <PGS_CSC.h>

PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];

PGSt_double Greenwich_Hour_Angle[2];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Define time requested */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );
time_offset[0] = 0.0;
time_offset[1] = 1.0;

/* Get Greenwich hour angles */

returnStatus = PGS_CSC_GreenwichHour( numValues, asciiUTC_A,
                time_offset, Greenwich_Hour_Angle );

/* Array Greenwich_Hour_Angle now contains the following values:

Greenwich_Hour_Angle[0] = 5.411645;  hours
Greenwich_Hour_Angle[1] = 5.411923;  hours

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_TD_3.f'

      INTEGER pgs_csc_greenwichhour

      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)

      DOUBLE PRECISION greenwich_hour_angle(2)

      INTEGER returnstatus
!
! Begin example
!

! Define time requested

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 1.0

! Get Greenwich hour angles

      returnstatus = pgs_csc_greenwichhour( numvalues, asciiutc_a,
     .          time_offset, greenwich_hour_angle )


! Array greenwich_hour_angle now contains the following values:

! greenwich_hour_angle(1) = 5.411645;  hours
! greenwich_hour_angle(2) = 5.411923;  hours
```

Notes:

A value PGSd_GEO_ERROR_VALUE may be returned in the variable *Greenwich_Hour_Angle* of the example. This indicates an error determining that value only; other elements of the output array are unaffected.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC

The physical reference to this file is defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to this file.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

11.2.11 PGS_CSC_nutate2000

Short explanation of what it's for: Nutate State Vector Between J2000 and Ephemeris Time (ET).

This function is in file: $PGSSRC/CBP/PGS_CSC_nutate2000.c

Examples:

Nutate a vector from J2000 to celestial coordinates of date..

C example:

```
#include <PGS_CSC.h>

     PGSt_SMF_status  returnStatus;
     PGSt_double      jedTDB[2];
     PGSt_double      dvnut[4];
     PGSt_double      posVel[6];

     jedTDB[0] = 2449720.5;
     jedTDB[1] = 0.25;

     posVel[0] =  6400000.0;
     posVel[1] = -5000000.0;
     posVel[2] =    40000.0;
     posVel[3] =     4000.0;
     posVel[4] =     7000.0;
     posVel[5] =    -6000.0;

   /*  get the nutation angles and rates */

    PGS_CSC_wahr2(jedTDB,dvnut);

    /* nutate the vector */

    returnStatus = PGS_CSC_nutate2000(6,jedTDB,dvnut,PGS_TRUE,posVel);

    /* The input vector "posVel" has been overwritten with the nutated value:

     posVel[0] =   6400276.14364
     posVel[1] =  -4999643.83137
     posVel[2] =     40334.16248
     posVel[3] =      3999.75622
     posVel[4] =      7000.00525
     posVel[5] =     -6000.15643
     */
```

FORTRAN example:

```
      implicit none
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'

      integer           pgs_csc_nutate2000
      integer           pgs_csc_wahr2

      integer           returnstatus
      integer           threeor6
      double precision  jedtdb(2)
      double precision  dvnut(4)
      double precision  frwd
      double precision  posvel(6)

      data jedtdb/2449720.5,0.25/
      data posvel/6400000.0,-5000000.0,40000.0,4000.0,7000.0,-6000.0/

      threeor6 = 6
      frwd = PGS_TRUE

!  get the nutation angles and rates

      returnstatus = pgs_csc_wahr2(jedtdb,dvnut)

!  nutate the vector

      returnstatus = pgs_csc_nutate2000(threeor6,jedtdb,dvnut,frwd,
     +                                  posvel)

!  the input vector "posvel" has been overwritten with the nutated value:

!    posVel(1) =   6400276.14364
!    posVel(2) =  -4999643.83137
!    posVel(3) =     40334.16248
!    posVel(4) =      3999.75622
!    posVel(5) =      7000.00525
!    posVel(6) =     -6000.15643
```

Notes:

Purpose: In the case of transforming from J2000, this function transforms a vector (position and velocity) after precession from J2000 to the correctly nutated coordinates -- i.e. the rotation (or Z) axis is along the Earth's angular velocity and the X axis is toward the equinox of date. (Precession gives the mean equinox of date and the program rotates a vector either to or from J2000, depending on the input flag.)

In the opposite case, in going from arbitrary epoch to J2000, this function nutates the vector to the "un-nutated" axis of date, after which it must be precessed to J2000 by the function PGS_CSC_precs2000().

This code was modified so it now takes either a 3 or 6 dimensional vector. When 6 dimensions are used, they must be in the order (position, velocity) because the transformation of velocity is slightly different. This function produces an output vector that overwrites the input vector. The code was kept this way to preserve its heritage. The user is cautioned that her/his input vector will therefore be altered by this function. The underlying rotation functions do not have this property.

11.2.12 PGS_CSC_ORBtoECI

Short explanation of what it's for: Convert a vector in Orbital (ORB) reference frame coordinates to Earth Centered Inertial (ECI) coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_ORBtoECI.c

Examples:

Two ORB vectors containing position are converted to two ECI vectors.
The first is a spacecraft ephemeris ECI vector in meters; the second is a unit vector.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double orb_vector[2][3];

PGSt_double eci_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

orb_vector[0][0] =  0.000; /* ORB X pos, meters */
orb_vector[0][1] =  0.000; /* ORB Y pos, meters */
orb_vector[0][2] =  0.000; /* ORB Z pos, meters */

orb_vector[1][0] =  0.228986;
orb_vector[1][1] = -0.545405;
orb_vector[1][2] =  0.806287;

/* Get ECI vector */

returnStatus = PGS_CSC_ORBtoECI( spacecraftID, numValues,
                       asciiUTC_A, time_offset, orb_vector,
                       eci_vector );

/* Matrix eci_vector now contains the following values:

eci_vector[0][0] =  1413531.574  ECI X pos, meters
eci_vector[0][1] = -6005427.214  ECI Y pos, meters
eci_vector[0][2] = -2693615.671  ECI Z pos, meters

eci_vector[1][0] = -0.153457      ECI unit vector
eci_vector[1][1] =  0.482829
eci_vector[1][2] =  0.862164

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_orbtoeci

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION orb_vector(3,2)

      DOUBLE PRECISION eci_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      orb_vector(1,1) =  0.000 ! ORB X pos, meters
      orb_vector(2,1) =  0.000 ! ORB Y pos, meters
      orb_vector(3,1) =  0.000 ! ORB Z pos, meters

      orb_vector(1,2) =  0.228986
      orb_vector(2,2) = -0.545405
      orb_vector(3,2) =  0.806287

! Get SC vector

      returnstatus = pgs_csc_orbtoeci( spacecraftid, numvalues,
     +                 asciiutc_a, time_offset, orb_vector,
     +                 eci_vector )


! Matrix eci_vector now contains the following values:

!      eci_vector(1,1) =  1413531.574   ECI X pos, meters
!      eci_vector(2,1) = -6005427.214   ECI Y pos, meters
!      eci_vector(3,1) = -2693615.671   ECI Z pos, meters

!      eci_vector(1,2) = -0.153457      ECI unit vector
!      eci_vector(2,2) =  0.482829
!      eci_vector(3,2) =  0.862164
```

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.13 PGS_CSC_ORBtoSC

Short explanation of what it's for: Convert a vector in Orbital (ORB) reference frame coordinates to Spacecraft (SC) reference frame coordinates .

This function is in file: $PGSSRC/CBP/PGS_CSC_ORBtoSC.c

Examples:

Two ORB vectors containing position are converted to two SC vectors.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double orb_vector[2][3];

PGSt_double sc_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

orb_vector[0][0] =  0.000; /* ORB X pos, meters */
orb_vector[0][1] =  0.000; /* ORB Y pos, meters */
orb_vector[0][2] =  0.000; /* ORB Z pos, meters */

orb_vector[1][0] =  0.228986;
orb_vector[1][1] = -0.545405;
orb_vector[1][2] =  0.806287;

/* Get SC vector */

returnStatus = PGS_CSC_ORBtoSC( spacecraftID, numValues,
                       asciiUTC_A, time_offset, orb_vector,
                       sc_vector );

/* Matrix sc_vector now contains the following values:

sc_vector[0][0] =  0.000     SC frame X pos, meters
sc_vector[0][1] =  0.000     SC frame Y pos, meters
sc_vector[0][2] =  0.000     SC frame Z pos, meters

sc_vector[1][0] =  0.228544  SC unit vector
sc_vector[1][1] = -0.548002
sc_vector[1][2] =  0.804649

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_orbtosc

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION orb_vector(3,2)

      DOUBLE PRECISION sc_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      orb_vector(1,1) =  0.000 ! ORB X pos, meters
      orb_vector(2,1) =  0.000 ! ORB Y pos, meters
      orb_vector(3,1) =  0.000 ! ORB Z pos, meters

      orb_vector(1,2) =  0.228986
      orb_vector(2,2) = -0.545405
      orb_vector(3,2) =  0.806287

! Get SC vector

      returnstatus = pgs_csc_orbtosc( spacecraftid, numvalues,
     +                 asciiutc_a, time_offset, orb_vector,
     +                 sc_vector )


! Matrix sc_vector now contains the following values:

! sc_vector(1,1) =  0.000    SC X pos, meters
! sc_vector(2,1) =  0.000    SC Y pos, meters
! sc_vector(3,1) =  0.000    SC Z pos, meters

! sc_vector(1,2) =  0.228544 SC unit vector
! sc_vector(2,2) = -0.548002
! sc_vector(3,2) =  0.804649
```

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.14 PGS_CSC_precs2000

Short explanation of what it's for: Precesses a vector between TDB Julian Date and J2000 Coordinates.

This function is in file: $PGSSRC/CBP/PGS_CSC_precs2000.c

Examples:

Precess a vector from J2000 to celestial coordinates of date..

C example:

```
#include <PGS_CSC.h>

     PGSt_SMF_status  returnStatus;
     PGSt_double      jedTDB[2];
     PGSt_double      posVel[6];

     jedTDB[0] = 2449720.5;
     jedTDB[1] = 0.25;

     posVel[0] =  6400000.0;
     posVel[1] = -5000000.0;
     posVel[2] =    40000.0;
     posVel[3] =     4000.0;
     posVel[4] =     7000.0;
     posVel[5] =    -6000.0;


    /* precess the vector */

    returnStatus = PGS_CSC_precs2000(6,jedTDB,PGS_TRUE,posVel);

    /* The input vector "posVel" has been overwritten with the nutated value:

     posVel[0] =    6394430.44572
     posVel[1] =   -5007144.69703
     posVel[2] =      36895.22797
     posVel[3] =       4004.90299
     posVel[4] =       6995.52993
     posVel[5] =      -6001.94250
     */
```

FORTRAN example:

```
      implicit none
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'

      integer           pgs_csc_precs2000
      integer           returnstatus
      integer           threeor6
      double precision  jedtdb(2)
      double precision  frwd
      double precision  posvel(6)

      data jedtdb/2449720.5,0.25/
      data posvel/6400000.0,-5000000.0,40000.0,4000.0,7000.0,-6000.0/

      threeor6 = 6
      frwd = PGS_TRUE

!  nutate the vector

      returnstatus = pgs_csc_nutate2000(threeor6,jedtdb,frwd,posvel)

!  the input vector "posvel" has been overwritten with the nutated value:

!    posVel(1) =   6394430.44572
!    posVel(2) =  -5007144.69703
!    posVel(3) =     36895.22797
!    posVel(4) =      4004.90299
!    posVel(5) =      6995.52993
!    posVel(6) =     -6001.94250
```

Notes:

This function is a simplified version of its precursor: PGS_CSC_precs3or6(). This function is specific to the case of precessing to or from the epoch of J2000. The various coefficients used are the constants that result for this epoch.

This function produces an output vector that overwrites the input vector. The code was kept this way to preserve its heritage. The user is cautioned that her/his input vector will be therefore be altered by this function. The underlying rotation functions do not have this property.

11.2.15 PGS_CSC_SCtoECI

Short explanation of what it's for: Convert a vector in Spacecraft (SC) reference frame coordinates to Earth Centered Inertial (ECI) coordinates .

This function is in file: $PGSSRC/CSC/PGS_CSC_SCtoECI.c

Examples:

Two SC vectors containing position are converted to two ECI vectors. The first is a spacecraft ephemeris SC vector in meters; the second is a unit vector (directional data).

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double time_offset[2];
PGSt_double sc_vector[2][3];

PGSt_double eci_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

sc_vector[0][0] =  0.000; /* SC X pos, meters */
sc_vector[0][1] =  0.000; /* SC Y pos, meters */
sc_vector[0][2] =  0.000; /* SC Z pos, meters */

sc_vector[1][0] =  0.228986; /* SC frame X direction cosine */
sc_vector[1][1] = -0.545405; /* SC frame Y direction cosine */
sc_vector[1][2] =  0.806287; /* SC frame Z direction cosine */

/* Get ECI vector */

returnStatus = PGS_CSC_SCtoECI( spacecraftID, numValues,
                       asciiUTC_A, time_offset, sc_vector,
                       eci_vector );

/* Matrix eci_vector now contains the following values:

eci_vector[0][0] =  1413531.574  ECI X pos, meters
eci_vector[0][1] = -6005427.214  ECI Y pos, meters
eci_vector[0][2] = -2693615.671  ECI Z pos, meters

eci_vector[1][0] = -0.153457     ECI X direction cosine
eci_vector[1][1] =  0.482829     ECI Y direction cosine
eci_vector[1][2] =  0.862164     ECI Z direction cosine


*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_sctoeci

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION time_offset(2)
      DOUBLE PRECISION sc_vector(3,2)

      DOUBLE PRECISION eci_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      sc_vector(1,1) =  0.000 ! SC X pos, meters
      sc_vector(2,1) =  0.000 ! SC Y pos, meters
      sc_vector(3,1) =  0.000 ! SC Z pos, meters

      sc_vector(1,2) =  0.228986 ! SC frame X direction cosine
      sc_vector(2,2) = -0.545405 ! SC frame Y direction cosine
      sc_vector(3,2) =  0.806287 ! SC frame Z direction cosine

! Get ECI vector

      returnstatus = pgs_csc_sctoeci( spacecraftid, numvalues,
     +                 asciiutc_a, time_offset, sc_vector,
     +                 eci_vector )


! Matrix eci_vector now contains the following values:

! eci_vector(1,1) =  1413531.574  ECI X pos, meters
! eci_vector(2,1) = -6005427.214  ECI Y pos, meters
! eci_vector(3,1) = -2693615.671  ECI Z pos, meters

! eci_vector(1,2) = -0.153457     ECI X direction cosine
! eci_vector(2,2) =  0.482829     ECI Y direction cosine
! eci_vector(3,2) =  0.862164     ECI Z direction cosine
```

Notes:

Aberration is taken into account in the transformation.

The input vector may be given in meters or may be a unit vector. If the input vector is not a unit vector, translation from earth center to spacecraft origin is accounted for.
No translation is done when the input vector is a unit vector.

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the ECIF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.16 PGS_CSC_SCtoORB

Short explanation of what it's for: Convert a vector in Spacecraft (SC) reference frame coordinates to Orbital (ORB) reference frame coordinates .

This function is in file: $PGSSRC/CSC/PGS_CSC_SCtoORB.c

Examples:

Two SC unit vectors containing position are converted to two ORB vectors.

C example:

```
#include <PGS_CSC.h>

PGSt_tag spacecraftID;
PGSt_integer numValues;
char asciiUTC_A[28];
PGSt_double UTC_offset[2];
PGSt_double sc_vector[2][3];

PGSt_double orb_vector[2][3];

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 0.0;

/* Fill input vectors */

sc_vector[0][0] =  0.000;     /* SC frame X pos, meters */
sc_vector[0][1] =  0.000;     /* SC frame Y pos, meters */
sc_vector[0][2] =  0.000;     /* SC frame Z pos, meters */

sc_vector[1][0] =  0.228544;  /* SC unit vector */
sc_vector[1][1] = -0.548002;
sc_vector[1][2] =  0.804649;


/* Get ORB vector */

returnStatus = PGS_CSC_SCtoORB( spacecraftID, numValues,
                       asciiUTC_A, UTC_offset, sc_vector,
                       orb_vector );

/* Matrix orb_vector now contains the following values:

orb_vector[0][0] =  0.000   ORB X pos, meters
orb_vector[0][1] =  0.000   ORB Y pos, meters
orb_vector[0][2] =  0.000   ORB Z pos, meters

orb_vector[1][0] =  0.228986
orb_vector[1][1] = -0.545405
orb_vector[1][2] =  0.806287

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_sctoorb

      INTEGER spacecraftid
      INTEGER numvalues
      CHARACTER*27 asciiutc_a
      DOUBLE PRECISION utc_offset(2)
      DOUBLE PRECISION sc_vector(3,2)

      DOUBLE PRECISION orb_vector(3,2)

      INTEGER returnstatus
!
! Begin example
!
! Assign spacecraft ID tag
!   PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 0.0

! Fill input vectors

      sc_vector(1,1) =  0.000     ! SC X pos, meters
      sc_vector(2,1) =  0.000     ! SC Y pos, meters
      sc_vector(3,1) =  0.000     ! SC Z pos, meters

      sc_vector(1,2) =  0.228544  ! SC unit vector
      sc_vector(2,2) = -0.548002
      sc_vector(3,2) =  0.804649

! Get ORB vector

      returnstatus = pgs_csc_sctoorb( spacecraftid, numvalues,
     +                 asciiutc_a, utc_offset, sc_vector,
     +                 orb_vector )


! Matrix orb_vector now contains the following values:

! orb_vector(1,1) =  0.000   ORB X pos, meters
! orb_vector(2,1) =  0.000   ORB Y pos, meters
! orb_vector(3,1) =  0.000   ORB Z pos, meters

! orb_vector(1,2) =  0.228986
! orb_vector(2,2) = -0.545405
! orb_vector(3,2) =  0.806287
```

Files:

This tool accesses the following files:

- leap seconds
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.17 PGS_CSC_PGS_CSC_SpaceRefract

Short explanation of what it's for: Estimate the refraction for a ray incident from space or a line of sight from space to the Earth's surface, based on the unrefracted zenith angle.

This function is in file: $PGSSRC/CBP/PGS_CSC_SpaceRefract.c

Examples:

Estimate the refraction for a ray incident from space.

C example

```
#include <PGS_CSC.h>

PGSt_SMF_status  returnStatus;

PGSt_double      spaceZenith;
PGSt_double      altitude;
PGSt_double      latitude;  /* not implemented at present */
PGSt_double      surfaceZenith;
PGSt_double      displacement;

/*
Begin example
*/

spaceZenith = 0.4;  /* radians */
altitude = 5000.0;  /* meters */

returnStatus = PGS_CSC_SpaceRefract(spaceZenith,altitude,latitude,
               &surfaceZenith,&displacement)

/* The following values are returned:

surfaceZenith =  0.3999259828  Refracted Zenith Angle
displacement  =  0.0000001245

*/
```

FORTRAN example

```
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_CSC_4.f'

      implicit none
      integer          pgs_csc_spacerefract
      integer          returnstatus

      double precision spacezenith
      double precision altitude
      double precision latitude   ! not implemented at present
      double precision surfacezenith
      double precision displacement

!
! Begin example
!
      spacezenith =  0.4     ! radians
      altitude    =  5000.0  ! meters

      returnstatus = pgs_csc_spacerefract(spacezenith,altitude,latitude,
     +                                    surfacezenith,displacement)

!    The following values are returned:

!    surfacezenith =  0.3999259828  Refracted Zenith Angle
!    displacement  =  0.0000001245
```

Notes:

This algorithm is intended as a mean-atmosphere approximation, valid for white light (for example, sunlight). Refraction is quite wavelength dependent, and in the atmosphere it will also depend strongly on local conditions (the weather, e.g.). The present algorithm is intended to be a reasonable approximation such that to do better one would need local and, for large zenith angles, regional weather.

The atmosphere model is used only to get the index of refraction at sea level. Latitude dependence is not implemented in the present version. Later, the sea level temperature and mean scale height will be altered to become functions of latitude.

11.2.18 PGS_CSC_SubSatPoint

Short explanation of what it's for: Determine where a vector to the spacecraft normal to the earth ellipsoid intersects the earth's surface. This point is called the sub-satellite point.
Velocity of this point is optionally determined along with the rate of change of altitude off the ellipsoid .

This function is in file: $PGSSRC/CBP/PGS_CSC_SubSatPoint.c

Examples:

Two intersection point coordinates and velocities are determined, one second of spacecraft ephemeris apart.

C example:

```
#include <PGS_CSC.h>

PGSt_tag        spacecraftID;
PGSt_integer    numValues;
char            asciiUTC_A[28];
PGSt_double     time_offset[2];
char            earthModel[21];
PGSt_boolean    velocity_flag;

PGSt_double     latitude[2];
PGSt_double     longitude[2];
PGSt_double     altitude[2];
PGSt_double     velocity[2][3];

PGSt_SMF_status returnStatus;

/*
Begin example
*/

/* Assign spacecraft ID tag
    PGSd_EOS_AM and PGSd_EOS_PM are also allowed */

      spacecraftID = PGSd_TRMM;

/* Define base time and offsets desired
   Base time is given in CCSDS ASCII Time code A format;
   CCSDS ASCII Time code B format is also allowed
   Offsets are in seconds */

numValues = 2;
strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z");
time_offset[0] = 0.0;
time_offset[1] = 1.0;

/* Define earth reference model */

strcpy( earthModel, "WGS84" );

/* Set velocity flag to PGS_FALSE if you do not need the
    velocity of the sub-satellite point */

velocity_flag = PGS_TRUE;

/* Get earth intersection point data */

returnStatus = PGS_CSC_SubSatPoint( spacecraftID, numValues,
               asciiUTC_A, time_offset, earthModel, velocity_flag,
               latitude, longitude, altitude, velocity );

/* The following values are returned:

latitude[0]  = -0.413986    Intersection pt latitude, radians
longitude[0] = -2.756803    Intersection pt longitude, radians
altitude[0]  =  357223.526  Distance from spacecraft to
                               intersection pt, meters

Velocity of the intersection pt on the ellipsoid, meters/sec:
velocity[0][0] =  3268.458  North component
velocity[0][1] =  6082.756  East component
velocity[0][2] =   -11.045  Rate of change of spacecraft
                              altitude relative to nadir

One second later:
latitude[1]  =   -0.413471
longitude[1] =   -2.755762
altitude[1]  =  357212.480
velocity[1][0] =  3271.369
velocity[1][1] =  6081.213
velocity[1][2] =   -11.046

*/
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_EPH_5.f'

      INTEGER pgs_csc_SubSatPoint

      INTEGER          spacecraftid
      INTEGER          numvalues
      CHARACTER*27     asciiutc_a
      DOUBLE PRECISION time_offset(2)
      CHARACTER*20     earthmodel
      INTEGER          velocity_flag

      DOUBLE PRECISION latitude(2)
      DOUBLE PRECISION longitude(2)
      DOUBLE PRECISION altitude(2)
      DOUBLE PRECISION velocity(3,2)

      INTEGER returnstatus

      INTEGER i,j
!
! Begin example
!
! Assign spacecraft ID tag
!    PGSd_EOS_AM and PGSd_EOS_PM are also allowed
!
      spacecraftid = PGSd_TRMM

! Define base time and offsets desired
!  Base time is given in CCSDS ASCII Time code A format;
!  CCSDS ASCII Time code B format is also allowed
!  Offsets are in seconds
!

     numvalues = 2
     asciiutc_a = '1998-06-30T10:51:28.320000Z'
     time_offset(1) = 0.0
     time_offset(2) = 1.0

! Define earth reference model

      earthModel = 'WGS84'

! Set velocity flag to PGS_FALSE if you do not need the
! velocity of the sub-satellite point

      velocity_flag = PGS_TRUE

! Get earth intersection point data

      returnstatus = pgs_csc_SubSatPoint( spacecraftid, numvalues,
     +         asciiutc_a, time_offset, earthmodel, velocity_flag,
     +         latitude, longitude, altitude, velocity )

! The following values are returned:

! latitude(1)  = -0.413986    Intersection pt latitude, radians
! longitude(1) = -2.756803    Intersection pt longitude, radians
! altitude(1)  =  357223.526  Distance from spacecraft to
!                                 intersection pt, meters

! Velocity of the intersection pt on the ellipsoid, meters/sec:
! velocity(1,1) = 3268.458  North component
! velocity(2,1) = 6082.756  East component
! velocity(3,1) =  -11.045  Rate of change of spacecraft
!                             altitude relative to nadir

! One second later:
! latitude(2)  =  -0.413471
! longitude(2) =  -2.755762
! altitude(2)  = 357212.480
! velocity(1,2) =  3271.369
! velocity(2,2) =  6081.213
! velocity(3,2) =   -11.046
```

Notes:

This function returns an error value if **any** of the input values are invalid. Returned values are all set to PGSd_GEO_ERROR_VALUE in this case.

The intersection of the vector in question with the Earth's equatorial plane defines the geodetic latitude.

Files:

This tool accesses the following files:

- leap seconds
- polar motion and UT1-UTC
- earth axis data
- spacecraft ephemeris/attitude

The physical references to these files are defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*. If you are using a PCF derived from that template, you need not do anything extra to enable access to these files.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

The exception is the spacecraft ephemeris/attitude file, which must be created by you for testing purposes at the SCF. Simulated files may be prepared through use of the *orbsim* utility; (sec. 7.1.2.1); alternatively, you may prepare them yourself (sec. 7.1.2.2).
This file must follow the ephemeris file naming convention, and must reside in directory *$PGSDAT/EPH*. This directory is specified in *$PGSRUN/PCF.v5*; individual spacecraft ephemeris/attitude filenames are not entered in the PCF.

11.2.19 PGS_CSC_wahr2

Short explanation of what it's for: Calculate Nutation Angles

This function is in file: $PGSSRC/CBP/PGS_CSC_wahr2.c

Examples:

Calculate Nutation Angles.

C example:

```
#include <PGS_CSC.h>

     PGSt_SMF_status  returnStatus;
     PGSt_double      jedTDB[2];
     PGSt_double      dvnut[4];

     jedTDB[0] = 2449720.5;
     jedTDB[1] = 0.25;

     /*  get the nutation angles and rates */

     PGS_CSC_wahr2(jedTDB,dvnut);


     /*  Array dvnut now contains the following values:

         dvnut[0] =  0.00006040835    Nutation in Longitude, radians
         dvnut[1] = -0.00003607640    Nutation in Obliquity, radians
         dvnut[2] =  0.00000000000333 Nutation rate in Longitude, radians/sec
         dvnut[3] =  0.00000000000259 Nutation rate in Obliquity, radians/sec

     */
```

FORTRAN example:

```
      implicit none
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC_4.f'

      integer           pgs_csc_wahr2
      integer           returnstatus
      double precision  jedtdb(2)
      double precision  dvnut(4)

      data jedtdb/2449720.5,0.25/

!  get the nutation angles and rates

      returnstatus = pgs_csc_wahr2(jedtdb,dvnut)

! Array dvnut now contains the following values:

!   dvnut(1) =  0.00006040835    Nutation in Longitude, radians
!   dvnut(2) = -0.00003607640    Nutation in Obliquity, radians
!   dvnut(3) =  0.00000000000333 Nutation rate in Longitude, radians/sec
!   dvnut(4) =  0.00000000000259 Nutation rate in Obliquity, radians/sec
```

11.2.20 PGS_CSC_PGS_CSC_ZenithAzimuth

Short explanation of what it's for: Determines zenith angle and azimuth of an arbitrary vector at a given geographic position. The vector may be either a look vector from the spacecraft to the earth, or the position vector of a celestial body .

This function is in file: $PGSSRC/CBP/PGS_CSC_ZenithAzimuth.c

Examples:

Two examples are given:
(1) For a single point on the earth, the zenith angle and azimuth of a spacecraft look vector are computed. Atmospheric refraction is accounted for. This example assumes that the example given in tool PGS_CSC_GetFOV_Pixel has been run first.
(2) The zenith angle of the Sun at a surface point is calculated. To get the Sun ECR input vector, tools PGS_CBP_Earth_CB_Vector and PGS_CSC_ECItoECR are called successively before the call to *PGS_CSC_ZenithAzimuth*.

C example 1: Zenith and azimuth of spacecraft look vector

```
#include <PGS_CSC.h>

PGSt_double ecr_vector[3];
PGSt_double latitude;
PGSt_double longitude;
PGSt_double altitude;
PGSt_tag vector_type;
PGSt_boolean zenith_only;
PGSt_boolean refraction;

PGSt_double zenith_angle;
PGSt_double azimuthal_angle;
PGSt_double refraction_decrease;

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/***** Data from the example output of PGS_CSC_GetFOV_Pixel *****/

/* Define spacecraft look vector in ECR frame. */

ecr_vector[0] = 0.20219599731;
ecr_vector[1] = 0.85458983269;
ecr_vector[2] = 0.47832310892;

/* Define earth location for which zenith and azimuthal
   angles desired */

latitude =  -0.392320;    /* geodetic latitude, radians */
longitude = -2.825456;    /*  longitude, radians */

/****** End data from PGS_CSC_GetFOV_Pixel ******/

/* Now set the altitude of surface point off the geoid, in
   meters (used only to estimate refraction, except in the
   case of the Moon, where it slightly affects the parallax)
   - user responsibility to provide this altitude (Toolkit
   provides DEM access in the AA tools) */

altitude =  0.0;  /* altitude of surface point, meters off geoid */

/* Indicate that ecr_vector is a look vector
   from the spacecraft.
   See Notes for other possible values. */

vector_type = PGSd_LOOK;

/* We want both zenith and azimuthal angles in this example */

zenith_only = PGS_FALSE;

/* Enable atmospheric refraction correction */

refraction = PGS_TRUE;

/* Get zenith and azimuthal angles */

returnStatus = PGS_CSC_ZenithAzimuth( ecr_vector, latitude,
                          longitude, altitude, vector_type,
                          zenith_only, refraction,
                          &zenith_angle, &azimuthal_angle,
                          &refraction_decrease );


/* The following values are returned:

zenith_angle        = 0.919446  refracted angle in radians
azimuthal_angle     = 1.912980  radians
refraction_decrease = 0.000381  decrease in zenith angle
                                 due to refraction, radians

*/
```

FORTRAN example 1: Zenith and azimuth of spacecraft look vector

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_CBP.f'

      INTEGER pgs_csc_zenithazimuth

      DOUBLE PRECISION ecr_vector(3)
      DOUBLE PRECISION latitude
      DOUBLE PRECISION longitude
      DOUBLE PRECISION altitude
      INTEGER vector_type
      INTEGER zenith_only
      INTEGER refraction

      DOUBLE PRECISION zenith_angle
      DOUBLE PRECISION azimuthal_angle
      DOUBLE PRECISION refraction_decrease

      INTEGER returnstatus
!
! Begin example
!

! **** Data taken from the example output of PGS_CSC_GetFOV_Pixel

! Define spacecraft look vector in ECR frame.

      ecr_vector(1) =  0.20219599731
      ecr_vector(2) =  0.85458983269
      ecr_vector(3) =  0.47832310892

! Define earth location for which zenith and azimuthal
! angles desired

      latitude  =  -0.392320  !    radians
      longitude =  -2.825456  !    radians

! **** End data from PGS_CSC_GetFOV_Pixel ******

      altitude = 0.0 ! altitude of surface point in meters off the geoid

! Indicate that ecr_vector is a look vector
!  from the spacecraft.
!  See Notes for other possible values.

      vector_type = PGSd_LOOK

! We want both zenith and azimuthal angles in this example

      zenith_only = PGS_FALSE

! Enable atmospheric refraction correction

      refraction = PGS_TRUE

! Get zenith and azimuthal angles

      returnStatus = pgs_csc_zenithazimuth( ecr_vector, latitude,
     +                     longitude, altitude, vector_type,
     +                     zenith_only, refraction,
     +                     zenith_angle, azimuthal_angle,
     +                     refraction_decrease );


! The following values are returned:

! zenith_angle        = 0.919446   refracted angle in radians
! azimuthal_angle     = 1.912980   radians
! refraction_decrease = 0.0003815  decrease in zenith angle
!                                   due to refraction, radians
```

C example 2: Zenith angle of Sun

```
#include <PGS_CSC.h>
/* Needed for other Toolkit calls in this example: */
#include <PGS_CBP.h>

char asciiUTC_A[28];
PGSt_double time_offset[1];
PGSt_double eci_vector_1[1][3];
PGSt_double eci_vector_2[1][6];
```

```
PGSt_double ecr_vector_2[1][6];

PGSt_double ecr_vector_1[3];
PGSt_double latitude;
PGSt_double longitude;
PGSt_double altitude;
PGSt_tag vector_type;
PGSt_boolean zenith_only;
PGSt_boolean refraction;

PGSt_double zenith_angle;
PGSt_double azimuthal_angle;
PGSt_double refraction_decrease;

PGSt_SMF_status returnStatus;
/*
Begin example
*/

/* First get ECI vector of Sun at the given time */

strcpy( asciiUTC_A, "1998-06-30T10:51:28.320000Z" );
time_offset[0] = 0.0;

returnStatus = PGS_CBP_Earth_CB_Vector( 1, asciiUTC_A,
                       time_offset, PGSd_SUN, eci_vector_1 );

/* Returned ECI vector in meters is
eci_vector_1[0][0] =  -22436733432.493786
eci_vector_1[0][1] =  138013995777.10355
eci_vector_1[0][2] =   59837305848.062561
*/

/* Next translate this vector to ECR coordinates */
/* First copy it over into the correct form */

eci_vector_2[0][0] = eci_vector_1[0][0];
eci_vector_2[0][1] = eci_vector_1[0][1];
eci_vector_2[0][2] = eci_vector_1[0][2];
eci_vector_2[0][3] = 0.0; /* velocity unused here */
eci_vector_2[0][4] = 0.0;
eci_vector_2[0][5] = 0.0;

returnStatus = PGS_CSC_ECItoECR( 1, asciiUTC_A,
                   time_offset, eci_vector_2, ecr_vector_2 );

/* Returned ECR vector in meters is
ecr_vector_2[0][0] = 132956286704.040
ecr_vector_2[0][1] =  43291706842.577
ecr_vector_2[0][2] =  59834999399.742
ecr_vector_2[0][3] = 0.0    velocity unused here
ecr_vector_2[0][4] = 0.0
ecr_vector_2[0][5] = 0.0
*/

/* Copy Sun ECR vector over into the correct form */
ecr_vector_1[0] = ecr_vector_2[0][0];
ecr_vector_1[1] = ecr_vector_2[0][1];
ecr_vector_1[2] = ecr_vector_2[0][2];

/* Define earth location for which zenith and azimuthal
   angles desired */

latitude =  -.547103859146;  /*   geodetic latitude, radians */
longitude = -.75014;         /*   longitude, radians */
altitude =  0.0;      /*  altitude of surface point, meters */

/* Define type of input ecr_vector
    (see Notes) */

vector_type = PGSd_SUN;

/* We want only zenith angle in this example */

zenith_only = PGS_TRUE;

/* Enable atmospheric refraction correction. */

refraction = PGS_TRUE;

/* Get zenith and azimuthal angles */

returnStatus = PGS_CSC_ZenithAzimuth( ecr_vector_1, latitude,
                          longitude, altitude, vector_type,
```

```
                          zenith_only, refraction,
                          &zenith_angle, &azimuthal_angle,
                          &refraction_decrease );


/* The following values are returned:

zenith_angle        = 1.392450  refracted angle in radians
azimuthal_angle     = 0.0
refraction_decrease = 0.001619  decrease in zenith angle
                                 due to refraction, radians

*/
```

FORTRAN example 2: Zenith angle of Sun

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_CSC.f'
      INCLUDE 'PGS_CSC_4.f'
      INCLUDE 'PGS_CBP.f'
! Needed for other Toolkit calls in this example:
      INCLUDE 'PGS_TD_3.f'
      INCLUDE 'PGS_CBP_6.f'

      INTEGER pgs_csc_zenithazimuth

      CHARACTER*28 asciiutc_a
      DOUBLE PRECISION time_offset(1)
      DOUBLE PRECISION eci_vector_1(3,1)
      DOUBLE PRECISION eci_vector_2(6,1)
      DOUBLE PRECISION ecr_vector_2(6,1)

      DOUBLE PRECISION ecr_vector_1(3)
      DOUBLE PRECISION latitude
      DOUBLE PRECISION longitude
      DOUBLE PRECISION altitude
      INTEGER vector_type
      INTEGER zenith_only
      INTEGER refraction

      DOUBLE PRECISION zenith_angle
      DOUBLE PRECISION azimuthal_angle
      DOUBLE PRECISION refraction_decrease

      INTEGER returnstatus
!
! Begin example
!

!  First get ECI vector of Sun at the given time

      asciiutc_a = '1998-06-30T10:51:28.320000Z'
      time_offset(1) = 0.0

      returnstatus = pgs_cbp_earth_cb_vector( 1, asciiutc_a,
     +                  time_offset, PGSd_SUN, eci_vector_1 )

!  Returned ECI vector in meters is
      eci_vector_1(1,1) = -22436733432.493786
      eci_vector_1(2,1) = 138013995777.10355
      eci_vector_1(3,1) =  59837305848.062561


!  Next translate this vector to ECR coordinates
!  First copy it over into the correct form

      eci_vector_2(1,1) = eci_vector_1(1,1)
      eci_vector_2(2,1) = eci_vector_1(2,1)
      eci_vector_2(3,1) = eci_vector_1(3,1)
      eci_vector_2(4,1) = 0.0 ! velocity unused here
      eci_vector_2(5,1) = 0.0
      eci_vector_2(6,1) = 0.0

      returnStatus = pgs_csc_ecitoecr( 1, asciiutc_a,
     .              time_offset, eci_vector_2, ecr_vector_2 )

!  Returned ECR vector in meters is
!       ecr_vector_2(1,1) = 132956286704.040
!       ecr_vector_2(2,1) = 43291706842.577
!       ecr_vector_2(3,1) = 59834999399.742
!       ecr_vector_2(4,1) = 0.0   velocity unused here
!       ecr_vector_2(5,1) = 0.0
!       ecr_vector_2(6,1) = 0.0
```

```
!  Copy Sun ECR vector over into the correct form
      ecr_vector_1(1) = ecr_vector_2(1,1)
      ecr_vector_1(2) = ecr_vector_2(2,1)
      ecr_vector_1(3) = ecr_vector_2(3,1)

! Define earth location for which zenith and azimuthal
! angles desired

      latitude  = -.547103859146 ! radians
      longitude = -.75014        ! radians
      altitude  = 0.0   ! altitude of surface point off geoid, in meters

! Define type of input ecr_vector
!    PGSd_MOON is also possible; other values are ignored
!    (see Notes)

      vector_type = PGSd_SUN

! We want only zenith angle in this example

      zenith_only = PGS_TRUE

! Enable atmospheric refraction correction

      refraction = PGS_TRUE

! Get zenith and azimuthal angles

returnStatus = pgs_csc_zenithazimuth( ecr_vector_1, latitude,
     +                     longitude, altitude, vector_type,
     +                     zenith_only, refraction,
     +                     zenith_angle, azimuthal_angle,
     +                     refraction_decrease );


! The following values are returned:

! zenith_angle        = 1.392450   refracted angle in radians
! azimuthal_angle     = 0.0
! refraction_decrease = 0.001619   decrease in zenith angle
!                                   due to refraction, radians
```

Notes:

Input vector must be in ECR reference frame coordinates.

5th argument in calling sequence *vector_type* may be one of the following:

PGSd_LOOKUse this for a spacecraft look vector. A special value is necessary here because the direction of this vector is opposite that of a celestial body vector.PGSd_MOONUse this if the celestial body in question is the Moon. In this case parallax is taken into account and the input ECR vector must be in meters (not a unit vector). The cb_id PGSd_MOON can be used for any near-Earth body (such as another spacecraft) ; it simply turns on the parallax correction based on the WGS84 ellipsoid and the altitude. For this purpose, altitude ought to be off the ellipsoid, but use altitude from the geoid if the refraction correction is turned on.PGSd_CB, or any other valid celestial body identifierUse this for any celestial body but the Moon. Parallax is not taken into account. Other celestial body identifiers are given in the Notes section of PGS_CBP_Earth_CB_Vector (sec. 11.2.2).

12. I/O Level 0 Access (IO_L0) Tools

12.1 I/O Level 0 Access (IO_L0) Tools Overview

12.1.1 Introduction

This section explains the usage of the I/O Level 0 data access tools.
These tools are mandatory for access to Level 0 data.

Level 0 data are raw science and engineering data received from either PACOR (for the TRMM spacecraft) or EDOS (for the EOS spacecrafts).

At the DAAC, these tools access L0 files previously staged by the Planning and Data Production Sub-system (PDPS), as specified in the plan you previously submitted.
At the SCF, you use either Toolkit modules or your own code to generate test L0 input files.
The explanations given here apply to usage in the testing environment at the SCF; many details of processing at the DAAC are not yet known, but these will be supplied where possible.

All of the data staged for a particular Application ID (APID) is considered by the Toolkit to be a "virtual data set". What this means to you is that even if there is more than one physical file staged for that APID, you have relatively seamless access to all of that data.

You may use this software to both generate test data files, and to access data in your production code.

12.1.2 Constructing a test Level 0 data file

There are two ways to do this:

- Use the interactive utility *L0sim*
- Call the Toolkit function PGS_IO_L0_File_Sim

The first method is to run from the unix command prompt

```
unix% $PGSBIN/L0sim
```

This utility will prompt you for input, such as file start and stop date, time interval between packets, APID, the name of a file containing simulated packet data, etc.
For TRMM instruments, it creates an SDPF-TRMM format main data file, plus an SFDU header file.
EOS AM and PM file formats are not yet known (except for packet formats); for now, for the file header, we provide part of the TRMM file header as a placeholder.

The second method involves your constructing a C or FORTRAN driver that calls PGS_IO_L0_File_Sim, which is the underlying function called by the utility described above. This may be useful if you want to customize your test file. The same files that are created by *L0sim* are created here.

Note: TRMM files have a "footer", which consists of quality and missing data unit information. The internal structure of the footer is neither simulated, nor is read access provided for it, in the TK5 version of this software.

12.1.3 Pseudo-code for Accessing L0 Data

Now that you have prepared your test input file, you are ready to read it. Below we provide pseudo-code which gives an overall view of how this is accomplished in your software.

```
Allocate memory for L0 data you will be saving

Call PGS_IO_L0_Open to get a virtual file handle,
       start and stop times of the available data

Determine the time range of the data you want to retrieve

If starting at some time other than the earliest time available
   Call PGS_IO_L0_SetStart to begin at the desired start time
End if

While still data in this virtual data set (APID) and/or time range

   Call PGS_IO_L0_GetHeader to retrieve header and footer
         information from the current physical file

   Unpack, save and/or process header and footer data

   While still packets in this physical file and/or time range

      Call PGS_IO_L0_GetPacket to read a single L0 packet

      Unpack, save and/or process packet data

   End while

End while

Call PGS_IO_L0_Close to close the virtual file
```

Notes:

The main Toolkit L0 functions return file header, footer and individual packet data in a character buffer. It is your responsibility to unpack this data.

The function PGS_IO_L0_GetHeader returns data from the physical Level 0 data file that is currently open. It is necessary to call it each time the end of a physical file is reached, if there is more than one such file per APID.

The pseudo-code shown is an example of processing a single Application ID (APID). Processing more than one APID could be done either consecutively (by looping over the given pseudo-code) or concurrently (by opening more than one virtual data set at once with PGS_IO_L0_Open).

The Toolkit functions and the example algorithm take into account the fact that the staged data for a single APID may consist of more than one physical file. This is not the case for TRMM, but may be for EOS AM and PM.

Footer information is returned for TRMM files only.
Also, TRMM processing includes a "housekeeping file", which consists of data for several non-science APIDs. This file is treated as a single virtual data set, just like science (single APID) files.

12.2 I/O Level 0 Access (IO_L0) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_IO_L0_* tools.

12.2.1 PGS_IO_L0_Close

Short explanation of what it's for: Close a virtual Level 0 data set.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_Close.c

Examples:

C example:

```
#include <PGS_IO.h>

PGSt_IO_L0_VirtualDataSet virtual_file;

PGSt_SMF_status returnStatus;

returnStatus = PGS_IO_L0_Close( virtual_file );
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'

      INTEGER pgs_io_l0_close

      INTEGER virtual_file

      INTEGER returnstatus

      returnstatus = pgs_io_l0_close( virtual_file )
```

Notes:

After this function is called, the currently open physical file is closed, and the internal table entries for this virtual data set are deleted.

Function PGS_IO_L0_Open must have been called before this tool is used.

12.2.2 PGS_IO_L0_File_Sim

Short explanation of what it's for: Create a simulated Level 0 data file for use at the SCF.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_File_Sim.c

Examples:

An EOS AM test file is generated, containing 1000 packets of different lengths and APIDs, starting at midnight June 1, 1999 and spaced at 1.024 second intervals.

C example:

```
#include <PGS_IO.h>

#define N 1000

PGSt_integer appID[N];
PGSt_integer firstPacketNum = 1;
char *startUTC = "1999-06-01T00:00:00";
PGSt_integer numValues = N;
PGSt_double  timeInterval = 1.024;
PGSt_integer dataLength[N];
PGSt_integer dummy1[2];
char *filename = "EOS_AM_L0_test";
char appData[350000];
PGSt_uinteger dummy2[2];
char *dummy3=NULL;
char *dummy4=NULL;

PGSt_integer i;

PGSt_SMF_status returnStatus;

/* Begin example */

/* Set APIDs and lengths of packet application data */
for (i=0;i<250;i++)
{
   dataLength[i] = 200;
   appID[i] = 80;
   dataLength[250+i] = 300;
   appID[250+i] = 81;
   dataLength[500+i] = 400;
   appID[500+i] = 82;
   dataLength[750+i] = 500;
   appID[750+i] = 83;
}
/* Fill appData buffer as desired here.
   Do not include packet header data; it is filled by this tool.
   Fill bytes 1-200 with first packet application data,
        bytes 201-400 with second packet application data, etc. */

/* Create simulated file */

returnStatus = PGS_IO_L0_File_Sim( EOS_AM,
                    appID, firstPacketNum, startUTC, numValues,
                    timeInterval, dataLength, dummy1, filename,
                    appData, dummy2, dummy3, dummy4 );


/* File EOS_AM_L0_test may now be used as input to
   other L0 Toolkit functions */
```

FORTRAN example:

```
        IMPLICIT NONE

        INCLUDE        'PGS_SMF.f'
        INCLUDE        'PGS_PC.f'
        INCLUDE        'PGS_PC_9.f'
        INCLUDE        'PGS_TD.f'
        INCLUDE        'PGS_IO.f'
        INCLUDE        'PGS_IO_1.f'

        INTEGER pgs_io_l0_file_sim

        INTEGER appid(1000)
        INTEGER firstpacketnum
        CHARACTER*27 startutc
        INTEGER numvalues
        INTEGER timeinterval
        INTEGER datalength(1000)
        INTEGER dummy1(2)
        CHARACTER*256 filename
        CHARACTER*350000 appdata
        INTEGER dummy2(2)
        CHARACTER*1 dummy3
        CHARACTER*1 dummy4

         INTEGER i
         INTEGER returnstatus

! Begin example

      firstpacketnum = 1
      startutc = '1999-06-01T00:00:00.000000'
      numvalues = 1000
      timeinterval = 1.024

!   Set APIDs and lengths of packet application data
      do i=1,250
         datalength(i) = 200
         appid(i) = 80
         datalength(250+i) = 300
         appid(250+i) = 81
         datalength(500+i) = 400
         appid(500+i) = 82
         datalength(750+i) = 500
         appid(750+i) = 83
      enddo


!   Fill appdata buffer as desired here.
!   Do not include packet header data; it is filled by this tool.
!   Fill bytes 1-200 with first packet application data,
!        bytes 201-400 with second packet application data, etc.


      filename = 'EOS_AM_L0_test'

      returnstatus = pgs_io_l0_file_sim( EOS_AM,
     +                        appid, firstpacketnum, startutc,
     +                        numvalues, timeinterval, datalength,
     +                        dummy1, filename, appdata,
     +                        dummy2, dummy3, dummy4 )

! File EOS_AM_L0_test may now be used as input to
!   other L0 Toolkit functions
```

Notes:

This tool is for use at the SCF only. It is not for use in production software.
It is being provided as a convenience to you; there is no Toolkit requirement for it or for *L0sim*.

The "dummy" arguments in the examples are used only for TRMM files.

For TRMM, the internal structure of the file footer (quality and missing data unit data) are not simulated.

Files:

This tool creates a simulated Level 0 data file of the specified name.
For TRMM, a Detached SFDU Header text file is also created.

12.2.3 PGS_IO_L0_GetHeader

Short explanation of what it's for: Read file header from the physical Level 0 data file currently open.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_GetHeader.c

Examples:

Data is read from a physical file header and (TRMM) file footer.

C example:

```
#include <PGS_IO.h>

#define HEADER_BUFFER_MAX 556
#define FOOTER_BUFFER_MAX 100000

PGSt_IO_L0_VirtualDataSet virtual_file;
PGSt_IO_L0_Header header_buffer[HEADER_BUFFER_MAX];
PGSt_IO_L0_Footer footer_buffer[FOOTER_BUFFER_MAX];

PGSt_SMF_status returnStatus;

returnStatus = PGS_IO_L0_GetHeader( virtual_file,
                    HEADER_BUFFER_MAX, header_buffer,
                    FOOTER_BUFFER_MAX, footer_buffer );

/* Unpack data returned in character buffer here */
```

FORTRAN example:

```
        IMPLICIT NONE

        INCLUDE        'PGS_SMF.f'
        INCLUDE        'PGS_PC.f'
        INCLUDE        'PGS_PC_9.f'
        INCLUDE        'PGS_TD.f'
        INCLUDE        'PGS_IO.f'
        INCLUDE        'PGS_IO_1.f'

        INTEGER pgs_io_l0_getheader

        INTEGER virtual_file
        CHARACTER*556 header_buffer
        CHARACTER*100000 footer_buffer
        INTEGER header_buffer_max
        INTEGER footer_buffer_max

        INTEGER returnstatus

        header_buffer_max = 556
        footer_buffer_max = 100000

        returnstatus = pgs_io_l0_getheader( virtual_file,
     +              header_buffer_max, header_buffer,
     +              footer_buffer_max, footer_buffer)

! Unpack data returned in character buffer here
```

Notes:

This function returns header and footer data in character (byte) buffers.
This data must be unpacked by the science software.

The last two arguments of this function are ignored for EOS AM and PM platforms.

Function PGS_IO_L0_Open must have been called before this tool is used.

Files:

This function accesses the Level 0 data file currently open for this virtual data set.

12.2.4 PGS_IO_L0_GetPacket

Short explanation of what it's for: Read a single CCSDS packet from a Level 0 data file.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_GetPacket.c

Examples:

Examples show how to read all packet data in one physical TRMM Level 0 data file.

C example:

```
#include <PGS_IO.h>

#define PACKET_BUFFER_MAX 7132

PGSt_IO_L0_VirtualDataSet virtual_file;
PGSt_IO_L0_Packet packet_buf[PACKET_BUFFER_MAX];

PGSt_integer packet_loop_flag;

PGSt_SMF_status returnStatus;

/* Call PGS_IO_L0_GetHeader here to read
      header and (TRMM) footer data */

packet_loop_flag = 1;
while( packet_loop_flag )
{
   returnStatus = PGS_IO_L0_GetPacket( virtual_file,
                         PACKET_BUFFER_MAX, packet_buf );

   if( returnStatus != PGS_S_SUCCESS )
   {
      if( returnStatus == PGSIO_M_L0_HEADER_CHANGED )
      {
          /* Out of data in this physical file,
             call PGS_IO_L0_GetHeader and continue reading. */

      }
      else if( returnStatus == PGSIO_W_L0_END_OF_VIRTUAL_DS )
      {
          /* Out of data in this virtual data set */
          packet_loop_flag = 0;
      }
      else
      {
         /* Error handling goes here */
      }
   }

   /* Unpack data returned in character buffer packet_buf here */

}
```

FORTRAN example:

```
      implicit none

      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'

      INTEGER pgs_io_l0_getpacket

      INTEGER virtual_file
      CHARACTER*7132 packet_buf

      INTEGER packet_loop_flag

      INTEGER returnstatus

      packet_loop_flag = 1
      do while( packet_loop_flag )
         returnStatus = pgs_io_l0_getpacket( virtual_file,
     +                     7132, packet_buf )
         if( returnStatus .ne. PGS_S_SUCCESS ) then
            if (returnstatus .eq. PGSIO_M_L0_HEADER_CHANGED) then
!              Out of data in this physical file,
!              call PGS_IO_L0_GetHeader and continue reading.

            else if (returnstatus .eq. PGSIO_W_L0_END_OF_VIRTUAL_DS) then
!              End of this virtual data set
               packet_loop_flag = 0
            else
!              Error handling goes here
            end if
         end if
!
!   Unpack data returned in character buffer packet_buf here
!

      end do
```

Notes:

This function returns packet data in character (byte) buffers.
This data must be unpacked by the science software.

The example shown is for TRMM, in which there is exactly one physical Level 0 data file for each virtual data set (Application ID). For EOS AM and PM, there might be more than one physical file per APID.
In this case, each time PGSIO_M_L0_HEADER_CHANGED is returned by PGS_IO_L0_GetPacket, your software should loop back to call PGS_IO_L0_GetHeader to read the new header information, then re-enter the packet read loop. When all physical files have been exhausted, PGS_IO_L0_GetPacket returns PGSIO_W_L0_END_OF_VIRTUAL_DS.

Function PGS_IO_L0_Open must have been called before this tool is used.

Files:

This function accesses the Level 0 data file currently open for this virtual data set.

12.2.5 PGS_IO_L0_Open

Short explanation of what it's for: Open a virtual Level 0 data set.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_Open.c

Examples:

A single LIS science APID (61) virtual data set is opened.
LIS (Lightning Imaging Sensor) is a TRMM instrument.

The examples assume the following entry in the Process Control File (PCF):

```
101|TRMM_G0091_1997-11-01T00:00:00Z_DATASET_V01_01||
          ||TRMM_G0091_1997-11-01T00:00:00Z_SFDU_V01_01|1
```

(Entry must appear on a single line in the PCF.)

C example:

```
#include <PGS_IO.h>

#define SCIENCE_FILE 101

PGSt_IO_L0_VirtualDataSet virtual_file;
PGSt_double start_time;
PGSt_double stop_time;

PGSt_SMF_status returnStatus;

/* Begin example */

returnStatus = PGS_IO_L0_Open( SCIENCE_FILE, PGSd_TRMM,
                     &virtual_file, &start_time, &stop_time);


/* Virtual file handle virtual_file may now be used as input
         to other L0 access tools
   start_time and stop_time now contain the start
         and stop times for all data staged for this APID,
         in TAI seconds since Jan. 1, 1993 */
```

go to Notes

FORTRAN example:

```
        IMPLICIT NONE

        INCLUDE        'PGS_SMF.f'
        INCLUDE        'PGS_PC.f'
        INCLUDE        'PGS_PC_9.f'
        INCLUDE        'PGS_TD.f'
        INCLUDE        'PGS_IO.f'
        INCLUDE        'PGS_IO_1.f'

        INTEGER        SCIENCE_FILE
        PARAMETER (SCIENCE_FILE=101)

        INTEGER pgs_io_l0_open

        INTEGER virtual_file
        DOUBLE PRECISION start_time
        DOUBLE PRECISION stop_time

        INTEGER returnstatus

! Begin example

        returnstatus = pgs_io_l0_open( SCIENCE_FILE,
     +          PGSd_TRMM, virtual_file, start_time, stop_time)

! Virtual file handle virtual_file may now be used as input
!         to other L0 access tools
! start_time and stop_time now contain the start
!         and stop times for all data staged for this APID,
!         in TAI seconds since Jan. 1, 1993
```

Notes:

For TRMM, there is only one physical file per APID per day. In this case each virtual data set (APID) corresponds to exactly one physical file.
For EOS AM and PM, there may be more than one physical file per APID (virtual data set).

In the Process Control File entry given in the example, the file name in the next-to-last field is the TRMM SFDU header file, which is a file that contains data associated with the given L0 file. This function does not open or access SFDU files; use functions PGS_IO_PC_GetFileAttr or PGS_IO_PC_GetFileByAttr to retrieve data from these files.

For the definition of the TAI time scale used, see sec. 9.1.2, Definition of Time Scales and Formats Used.

Files:

This tool provides access to all staged Level 0 files for the given PCF logical ID, which normally means a given Application ID (APID). A logical ID may also be used for a TRMM housekeeping file, which contains many APIDs.

See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

12.2.6 PGS_IO_L0_SetStart

Short explanation of what it's for: Set start time for reading staged Level 0 data.

This function is in file: $PGSSRC/IO/L0/PGS_IO_L0_SetStart.c

Examples:

Examples show how to start processing 20 minutes after the start of the Level 0 data.
The value of variable *start_time* is assumed as returned from PGS_IO_L0_Open.

C example:

```
#include <PGS_IO.h>

PGSt_IO_L0_VirtualDataSet virtual_file;
PGSt_double start_time;

PGSt_SMF_status returnStatus;

returnStatus = PGS_IO_L0_SetStart(virtual_file, start_time+1200.0);

/* The virtual file pointer is now set to the desired time */
```

FORTRAN example:

```
      IMPLICIT NONE
      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_PC.f'
      INCLUDE 'PGS_PC_9.f'
      INCLUDE 'PGS_TD.f'
      INCLUDE 'PGS_IO.f'
      INCLUDE 'PGS_IO_1.f'

      INTEGER pgs_io_l0_setstart

      INTEGER virtual_file
      DOUBLE PRECISION start_time
      DOUBLE PRECISION new_start_time

      INTEGER returnstatus

      new_start_time = start_time+1200.0

      returnstatus = pgs_io_l0_setstart( virtual_file,
     +                                 new_start_time )

! The virtual file pointer is now set to the desired time
```

Notes:

Function PGS_IO_L0_Open must have been called before this tool is used.

Files:

This function accesses one or more staged Level 0 data files.

13. Constants and Unit Conversion (CUC) Tools

13.1 Constants and Unit Conversion (CUC) Tools Overview

13.1.1 Introduction

The tools in this section are used to access constants and unit conversion data. i.e., data required for production processing which is obtained from independent standardized external sources.

These tools are optional, in the sense that you may use your own functions to access this data or hard code the values if you so desire.

13.1.2 Accessing Constants data

There is one Toolkit function which accesses Constants data, PGS_CUC_Cons. This function reads the value of a constant either from an official list of constants supplied by the EOS Project Science Office, or from a file that you have constructed on your own.
As the official list has not been determined at this writing (March 1995), a dummy test file is included in the current Toolkit delivery.

13.1.3 Accessing Conversion data

There is one Toolkit function which accesses Conversion data, PGS_CUC_Conv. This function is used by the user to retrieve the relevant conversion data to convert between two named units.

This function uses the freeware UdUnits software package from the University Corporation for Atmospheric Research (UCAR).

13.2 Constants and Unit Conversion (CUC) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_CUC_* tools.

13.2.1 PGS_CUC_Cons

Short explanation of what it's for: Retrieve the value of a constant from either a NASA-approved file or from your own file.

This function is in file: $PGSSRC/CUC/PGS_CUC_Cons.c

Examples:

Retrieve the value of test parameter "H" from the NASA-approved file.

Relevant to this example, the dummy test file contains the line

```
H = 31.567
```

C example:

```
#include <PGS_CUC.h>
#define OFFICIAL_CONSTANTS 10999  /* This never changes */

char *parm = "H";

PGSt_double h;

PGSt_SMF_status returnStatus;

returnStatus = PGS_CUC_Cons(OFFICIAL_CONSTANTS, parm, &h);

/* Variable h now contains the value 31.567 */
```

FORTRAN example:

```
      IMPLICIT NONE

      INCLUDE 'PGS_CUC_11.f'
      INCLUDE 'PGS_SMF.f'

      integer OFFICIAL_CONSTANTS
      parameter (OFFICIAL_CONSTANTS=10999)

      integer pgs_cuc_cons

      character*80 parm

      double precision h

      integer returnstatus

      parm = 'h'

      returnstatus = pgs_cuc_cons(OFFICIAL_CONSTANTS, parm, h)

C Variable h now contains the value 31.567
```

Notes:

For purposes of testing this tool, a dummy test file has been included in the current Toolkit delivery.

You may construct your own constants file for use by this function.
It need only be in PARAMETER = VALUE format, and defined in the Process Control file. (Note that IDs 10000 to 10999 are reserved for Toolkit use.)
Your file becomes part of the delivery of your PGE to the DAAC.

Files:

This tool accesses the following file:

- Dummy test constants file *PGS_CUC_maths_parameters*

The physical reference to this file is defined in the Process Control File (PCF) template supplied with the Toolkit, *$PGSRUN/PCF.v5*.
Its logical file ID in the PCF is 10999. Use this value in your code to enable access to this file.
See sec. 3.1.2, Constructing your Process Control file, for information about PCF entries.

13.2.2 PGS_CUC_Conv

Short explanation of what it's for: Retrieve parameters needed for units conversion.

This function is in file: $PGSSRC/CUC/PGS_CUC_Conv.c

Examples:

Convert 0.85 atmospheres to bars.

C example:

```
#include <PGS_CUC.h>

char inpUnit[30];
char outUnit[30];

PGSt_double outSlope;
PGSt_double outIntercept;

PGSt_double press_atm;
PGSt_double press_bar;

PGSt_SMF_status returnStatus;

strcpy( inpUnit, "atm" );
strcpy( outUnit, "bar" );

/* Call Toolkit function to find the conversion parameters */

returnStatus = PGS_CUC_Conv(inpUnit, outUnit,
                    &outSlope, &outIntercept);

/* Variable outSlope now contains the value 1.013250 bar/atm */
/* Variable outIntercept now contains the value 0.000000 bar */

press_atm = 0.85;

press_bar = outSlope * press_atm;

/* Variable press_bar now contains the value 0.861262 bars */
```

FORTRAN example:

```
      IMPLICIT NONE

      INCLUDE 'PGS_CUC_11.f'
      INCLUDE 'PGS_SMF.f'

      integer pgs_cuc_conv

      character*30 inpunit
      character*30 outunit

      double precision outslope
      double precision outintercept

      double precision press_atm
      double precision press_bar

      integer returnstatus

      inpunit = 'atm'
      outunit = 'bar'

      returnstatus = pgs_cuc_conv( inpunit, outunit,
     .                         outslope, outintercept )

/* Variable outslope now contains the value 1.013250 bar/atm */
/* Variable outintercept now contains the value 0.000000 bar */

      press_atm = 0.85D0

      press_bar = outslope * press_atm;

/* Variable press_bar now contains the value 0.861262 bars */
```

Notes:

This function makes use of the UdUnits ("Unidata Units") freeware package from UCAR.

Files:

This tool accesses the following file:

- Units conversion file *udunits.dat*

By default, this file is put in directory $PGSSRC/CUC/UDUNITS when the Toolkit is installed.

14. Geocoordinate Transformation (GCT) Tools

14.1 Geocoordinate Transformation (GCT) Tools Overview

14.1.1 Introduction

These tools provide an interface to initialize and perform Geocoordinate Transformations in the forward and inverse directions.

This software is based on the projections provided in the GCTP geo-coordinate transformation package provided by USGS.

14.1.2 Initialization of the tool

PGS_GCT_Init must be used first, whenever you want to do a new kind of geocoordinate transformation. Each projection requires a number of parameters to be initialized prior to use; this function provides the generic interface to perform these initializations.

14.1.3 Geocoordinate Transformations

There is one Toolkit function which performs the geocoordinate transformations, PGS_GCT_Proj. This Tool provides a general interface to perform Geocoordinate Transformations in the forward/inverse directions, ie. from geographical coordinates (latitude , longitude) to Cartesian coordinates (x, y) of the given projection.
Access to all projections defined in the GCTP package is provided. A list of these projections is given in the PGS_GCT_Init Notes.

You **must** call PGS_GCT_Init prior to using this function.
The tool may be used to perform the same transformation several times, for the same parameters. If the same projection is used with different parameters. PGS_GCT_Init must be called again.

This tool is so constructed so that new projections may be added easily.

14.2 Geocoordinate Transformation (GCT) Tool Descriptions

This section contains an alphabetical listing of the descriptions of the individual PGS_GCT_* tools.

14.2.1 PGS_GCT_Init

Short explanation of what it's for: Initializes the Toolkit for a given geo-coordinate projection and direction (forward or reverse).

This function is in file: $PGSSRC/CUC/PGS_GCT_Init.c

Examples:

Examples initialize for polar stereographic projection transformations.

C example:

```
#include <PGS_GCT.h>

PGSt_integer projId;
PGSt_double projParam[13];
PGSt_integer directFlag;

PGSt_integer i;

PGSt_integer returnStatus;

/* Define projection ID for Polar Sterographic projection.
   List of projections is given in the Notes */

projId = PS;

/* Define input parameters. */

for (i=0;i<13;i++) projParam[i] = 0.0;

/* Define axes of earth ellipsoid (meters) */

projParam[0] = 6378137.0;
projParam[1] = 6356752.3;

/* Define longitude down below pole of map (radians) */

projParam[4] = -0.2;

/* Define latitude of true scale (radians) */

projParam[5] = 1.0;

/* Define false easting and northing (meters) */

projParam[6] = 0.0;
projParam[7] = 0.0;

/* Define direction of transformation: lat/long to map coords
   (PGSd_GCT_INVERSE gives the reverse transformation) */

directFlag = PGSd_GCT_FORWARD;

/* Initialize for Polar Stereographic projection
   transformations */

returnStatus = PGS_GCT_Init(projId, projParam, directFlag);

/* You may now call PGS_GCT_Proj to do transformations */
```

Fortran example:

IMPLICIT NONE INCLUDE 'PGS_GCT.f' INCLUDE 'PGS_GCT_12.f' INCLUDE 'PGS_SMF.f' integer pgs_gct_init integer projid double precision projparam(13) integer directflag integer i integer returnstatus C Define projection ID for Polar Sterographic projection. C List of projections is given in the Notes projid = PS C Define input parameters. do 10 i=1,13 projparam(i) = 0.0 10 continue C Define axes of earth ellipsoid (meters) projparam(1) = 6378137.0 projparam(2) = 6356752.3 C Define longitude down below pole of map (radians) projparam(5) = -0.2 C Define latitude of true scale (radians) projparam(6) = 1.0 C Define false easting and northing (meters) projparam(7) = 0.0 projparam(8) = 0.0 C Define direction of transformation: lat/long to map coords C (PGSd_GCT_INVERSE gives the reverse transformation) directflag = PGSd_GCT_FORWARD C Initialize for Polar Stereographic projection C transformations returnstatus = **pgs_gct_init**(projid, projparam, directflag) C You may now call pgs_gct_proj to do transformations

Notes:

This routine simply initializes the parameters required by a particular projection.
Actual transformations are done by PGS_GCT_Proj.

New projections may be added if desired.

IMPORTANT: All blank array elements must be set to zero by you.

Projection IDs (Name in parentheses)

- UTM (Universal Transverse Mercator)
- SPCS (State Plane Coordinates)
- ALBERS (Albers Conical Equal Area)
- LAMCC (Lambert Conformal Conic)
- MERCAT (Mercator)
- PS (Polar Stereographic)
- POLYC (Polyconic)
- EQUIDC (Equidistant Conic)
- TM (Transverse Mercator)
- STEREO (Stereographic)

- LAMAZ (Lambert Azimuthal Equal Area)
- AZMEQD (Azimuthal Equidistant)
- GNOMON (Gnomonic)
- ORTHO (Orthographic)
- GVNSP (General Vertical Near-Side Perspective)
- SNSOID (Sinusoidal)
- EQRECT (Equirectangular)
- MILLER (Miller Cylindrical)
- VGRINT (Van der Grinten)
- HOM (Hotine Oblique Mercator--HOM)
- ROBIN (Robinson)
- SOM (Space Oblique Mercator--SOM)
- ALASKA (Modified Stereographic Conformal-- Alaska)
- GOOD (Interrupted Goode Homolosine)
- MOLL (Mollweide)
- IMOLL (Interrupted Mollweide)
- HAMMER (Hammer)
- WAGIV (Wagner IV)
- WAGVII (Wagner VII)
- OBLEQA (Oblated Equal Area)

14.2.2 PGS_GCT_Proj

Short explanation of what it's for: Perform a transformation for the geo-coordinate projection inititalized by PGS_GCT_Init.

This function is in file: $PGSSRC/CUC/PGS_GCT_Proj.c

Examples:

Examples perform polar stereographic projection transformations.
We assume that the example for PGS_GCT_Init has been run first.

C example:

```
#include <PGS_GCT.h>

PGSt_integer projId;
PGSt_integer directFlag;
PGSt_integer nPts;
PGSt_double longitude[2];
PGSt_double latitude[2];

PGSt_double mapX[2];
PGSt_double mapY[2];
PGSt_integer dummy[2];

PGSt_integer returnStatus;

/* Define projection ID for Polar Sterographic projection.
   PGS_GCT_Init must have been called previously
   with the same value.
   List of projections is given in the PGS_GCT_Init Notes */

projId = PS;

/* Define direction of transformation: lat/long to map coords
   PGS_GCT_Init must have been called previously
   with the same value. */

directFlag = PGSd_GCT_FORWARD;

/* Define lat/long (radians) for which to find map coordinates */

nPts = 2;
longitude[0] = 1.4;
latitude[0] = 0.2;
longitude[1] = -1.4;
latitude[1] = 0.2;

/* Transform from lat/long to map coords */

returnStatus = PGS_GCT_Proj(projId, directFlag, nPts,
                    longitude, latitude, mapX, mapY, dummy);

/* Variables mapX and mapY now contain the following values:

mapX[0] = 9580513.1963976845 meters
mapX[1] = 279865.7426374513 meters

mapY[0] = -8933221.8612986654 meters
mapY[1] = -3473054.1483063293 meters

*/
```

Fortran example:

```
      IMPLICIT NONE

      INCLUDE 'PGS_GCT.f'
      INCLUDE 'PGS_GCT_12.f'
      INCLUDE 'PGS_SMF.f'

      integer pgs_gct_proj

      integer projid
      integer directflag
      integer nPts
      double precision longitude(2)
      double precision latitude(2)

      double precision mapx(2)
      double precision mapy(2)
      double precision dummy(2)

      integer returnstatus


C Define projection ID for Polar Sterographic projection.
C PGS_GCT_Init must have been called previously
C with the same value.
C List of projections is given in the Notes

      projid = PS

C Define direction of transformation: lat/long to map coords
C PGS_GCT_Init must have been called previously
C with the same value.

      directflag = PGSd_GCT_FORWARD

C Define lat/long (radians) for which to find map coordinates

      npts = 2
      longitude(1) = 1.4
      latitude(1) = 0.2
      longitude(2) = -1.4
      latitude(2) = 0.2

C Transform from lat/long to map coords

      returnstatus = pgs_gct_proj(projid, directflag, npts,
     .             longitude, latitude, mapx, mapy, dummy)

C Variables mapX and mapY now contain the following values:

C mapX(1) = 9580513.1963976845 meters
C mapX(2) = 279865.7426374513 meters

C mapY(1) = -8933221.8612986654 meters
C mapY(2) = -3473054.1483063293 meters
```

Notes:

Function PGS_GCT_Init must have been called before this function is called.

Each time you want to change projections, or if you want to reverse the direction (from forward to reverse or vice-versa), you must call PGS_GCT_Init again first.

Variable *dummy* used in the examples is used only for the UTM transformation, for zone number.

Appendix A. Sample Status Message Text File

```
##############################################################
# BEGIN_FILE_PROLOG:
#
# FILENAME:
#   AVHRR.t Return code definitions for AVHRR
#           (SMF seed value 99)
#
#
# DESCRIPTION:
#
#   This file contains PGS_SMF standard return code
#   definitions for the AVHRR code.
#
#   The file is intended to be used as input by the smfcompile
#   utility, which generates the PGS_99 message file, and the
#   PGS_PATHFINDER_99.h header
#   file.
#
# AUTHOR:
#   Tom Atwater
#
# HISTORY:
#  19_Sep-1994 TWA Initial version
#
# END_FILE_PROLOG:
#############################################################
%INSTR    = AVHRR
%LABEL    = PATHFINDER
%SEED     = 99
#
# messages for all AVHRR code
#
PATHFINDER_F_OPEN_BIN_OUT_FILE
                         FATAL_ERROR...error opening %s
PATHFINDER_F_OPEN_ANC_FILE
                         FATAL ERROR...%s
PATHFINDER_F_MEM_ALLOC_FAIL
                         FATAL ERROR... allocating memory for %s
PATHFINDER_E_EPH_MEM_ALLOC_FAIL
                         Error %s
PATHFINDER_F_OPEN_BINARY_FILE
                         FATAL_ERROR...error opening binary file
PATHFINDER_W_CANT_WRITE_LOG
                         WARNING: Can't write to log file
PATHFINDER_F_NUM_GAC_FILES
                         FATAL_ERROR...determining no. gac files
PATHFINDER_W_CLOSE_GAC_FILE
                         WARNING...could not close last file
PATHFINDER_F_OPEN_GAC_FILE
                         FATAL_ERROR...Can't open gac file
PATHFINDER_F_OPEN_PROCLOG_FILE
                         FATAL_ERROR...error creating log file
PATHFINDER_W_READING_PC_FILE
                         FATAL_ERROR...reading PC file: %s
PATHFINDER_W_READ_REQ_SIZE_X
                         WARNING: Error reading requested size x
PATHFINDER_W_READ_REQ_SIZE_Y
                         WARNING: Error reading requested size x
PATHFINDER_W_READ_WAIT_TIME
                         WARNING: Error reading wait time
PATHFINDER_F_OPEN_BIN_OUT_FILE
                         FATAL_ERROR...error opening %s
PATHFINDER_F_PROC_INIT_ERROR
                         FATAL_ERROR...%s
PATHFINDER_W_NO_LOG_FILES
                         WARNING: Problem sending log files
PATHFINDER_W_NO_PROC_LOG WARNING: Problem sending GSFC log file
PATHFINDER_N_PROCESSING_DONE
                         SUCCESS: AVHRR complete at %s
```

Appendix B. Sample Process Control File (PCF)

Entries specific to the Pathfinder AVHRR/Land example in this Primer appear in **bold**.

```
# Process Control File: Pathfinder AVHRR/Land Toolkit Prototype
#
# Environment variable PGS_PC_INFO_FILE must point to this file
#
```

```
?   SYSTEM RUNTIME PARAMETERS
# -----------------------------------------------------------------
# Production Run ID - unique production instance identifier
# -----------------------------------------------------------------
1
# -----------------------------------------------------------------
# Software ID - unique software configuration identifier
# -----------------------------------------------------------------
1
#
?   PRODUCT INPUT FILES
# [ next line is for default location ]
! ~/runtime
#
# -----------------------------------------------------------------
# Pathfinder AVHRR/Land input files
# -----------------------------------------------------------------
201|87002002709.no9_gac|||||1
401|goldtopolandsea8.bin|||||1
402|gridtoms_1987_sngl_ntwk|||||1
403|ephem8788.dat|||||1
404|timecorr8788.dat|||||1
405|SDSannotations.dat|||||1
406|HDFmetadata.dat|||||1
410|jan021987.proclog|||||1
#
#
# -----------------------------------------------------------------
# Toolkit product input files
# -----------------------------------------------------------------
#
# -----------------------------------------------------------------
# These are actual ancillary data set files - supplied by ECS or
# the user.
# The following are supplied for purposes of tests and as a
# useful set of ancillary data.
# The files will be located in $PGSHOME/runtime.
#
# WARNING! DO NOT MODIFY DEFAULT FILE LOCATION FOR THIS SECTION
# unless you have relocated these data set files to the location
# specified by the location's new setting.
# -----------------------------------------------------------------
10780|usatile12||||10751|12
10780|usatile11||||10750|11
10780|usatile10||||10749|10
10780|usatile9||||10748|9
10780|usatile8||||10747|8
10780|usatile7||||10746|7
10780|usatile6||||10745|6
10780|usatile5||||10744|5
10780|usatile4||||10743|4
10780|usatile3||||10742|3
10780|usatile2||||10741|2
10780|usatile1||||10740|1
10951|mowe13a.img|||||1
10952|owe13a.img|||||1
10953|owe14d.img|||||1
10954|owe14dr.img|||||1
10955|etop05.dat|||||1
10956|fnocazm.img|||||1
10957|fnococm.img|||||1
10958|fnocpt.img|||||1
10959|fnocrdg.img|||||1
10960|fnocst.img|||||1
10961|fnocurb.img|||||1
10962|fnocwat.img|||||1
10963|fnocmax.imgs|||||1
10964|fnocmin.imgs|||||1
10965|fnocmod.imgs|||||1
10966|srzarea.img|||||1
10967|srzcode.img|||||1
10968|srzphas.img|||||1
10969|srzslop.img|||||1
10970|srzsoil.img|||||1
10971|srztext.img|||||1
10972|nmcRucPotPres.datrepack|||||1
10973|tbase.bin||||10915|1
10974|tbase.br||||10919|4
10974|tbase.bl||||10918|3
10974|tbase.tr||||10917|2
10974|tbase.tl||||10916|1
# -------------------------------------------------------------------------
# Constant & Unit Conversion file
# IMPORTANT NOTE: THIS FILE WILL BE SUPPLIED AFTER TK4 DELIVERY!
```

```
# -----------------------------------------------------------------------
10999|PGS_CUC_maths_parameters|||||1
#
#
# -----------------------------------------------------------------------
# The following are for the PGS_GCT tool only.
# The IDs are #defined in the PGS_GCT.h file
# -----------------------------------------------------------------------
10200|nad27sp|~/runtime||||1
10201|nad83sp|~/runtime||||1
# ------------------------------------------------------------
# The following are for the PGS_AA_DCW tool only.
# The IDs are #defined in the PGS_AA_DCW.h file
# ------------------------------------------------------------
10990|eurnasia/|||||1
10991|noamer/|||||1
10992|soamafr/|||||1
10993|sasaus/|||||1
#
# ------------------------------------------------------------
# End Toolkit product input files
# ------------------------------------------------------------
#
?   PRODUCT OUTPUT FILES
# [ next line is for default location ]
! ~/runtime
#
# ------------------------------------------------------------
# Pathfinder AVHRR/Land main output file
# ------------------------------------------------------------
301|test11.hdf|||||1
#
?   SUPPORT INPUT FILES
# [ next line is for default location ]
! ~/supportinput
#
# ------------------------------------------------------------
# Toolkit support input files
# ------------------------------------------------------------
#
# ------------------------------------------------------------
# This ID is #defined in PGS_AA_Tools.h . This file contains
# the IDs for all support and format files shown.
# ------------------------------------------------------------
10900|indexFile|~/runtime||||1
#
# ------------------------------------------------------------
# These are support files for the data set files - to be created
#  by user (not necessarily a one-to-one relationship)
# The IDs must correspond to the logical IDs in the index file
# ------------------------------------------------------------
10901|mowe13aSupport|~/runtime||||1
10902|owe13aSupport|~/runtime||||1
10903|owe14Support|~/runtime||||1
10904|etop05Support|~/runtime||||1
10905|fnoc1Support|~/runtime||||1
10906|fnoc2Support|~/runtime||||1
10907|zobler1Support|~/runtime||||1
10908|zobler2Support|~/runtime||||1
10909|nmcRucSupport|~/runtime||||1
10915|tbaseSupport|~/runtime||||1
10916|tbase1Support|~/runtime||||1
10917|tbase2Support|~/runtime||||1
10918|tbase3Support|~/runtime||||1
10919|tbase4Support|~/runtime||||1
10740|usatile1Support|~/runtime||||1
10741|usatile2Support|~/runtime||||1
10742|usatile3Support|~/runtime||||1
10743|usatile4Support|~/runtime||||1
10744|usatile5Support|~/runtime||||1
10745|usatile6Support|~/runtime||||1
10746|usatile7Support|~/runtime||||1
10747|usatile8Support|~/runtime||||1
10748|usatile9Support|~/runtime||||1
10749|usatile10Support|~/runtime||||1
10750|usatile11Support|~/runtime||||1
10751|usatile12Support|~/runtime||||1
#
# ------------------------------------------------------------
# The following are format files for each data set file
# (not necessarily a one-to-one relationship)
# The IDs must correspond to the logical IDs in the index file
# ------------------------------------------------------------
10920|mowe13a.bfm|~/runtime||||1
```

```
10921|owe13a.bfm|~/runtime||||1
10922|owe14d.bfm|~/runtime||||1
10923|owe14dr.bfm|~/runtime||||1
10924|etop05.bfm|~/runtime||||1
10925|fnocAzm.bfm|~/runtime||||1
10926|fnocOcm.bfm|~/runtime||||1
10927|fnocPt.bfm|~/runtime||||1
10928|fnocRdg.bfm|~/runtime||||1
10929|fnocSt.bfm|~/runtime||||1
10930|fnocUrb.bfm|~/runtime||||1
10931|fnocWat.bfm|~/runtime||||1
10932|fnocMax.bfm|~/runtime||||1
10933|fnocMin.bfm|~/runtime||||1
10934|fnocMod.bfm|~/runtime||||1
10935|srzArea.bfm|~/runtime||||1
10936|srzCode.bfm|~/runtime||||1
10937|srzPhas.bfm|~/runtime||||1
10938|srzSlop.bfm|~/runtime||||1
10939|srzSoil.bfm|~/runtime||||1
10940|srzText.bfm|~/runtime||||1
10941|nmcRucSigPotPres.bfm|~/runtime||||1
10942|tbase.bfm|~/runtime||||1
10943|tbase1.bfm|~/runtime||||1
10944|tbase2.bfm|~/runtime||||1
10945|tbase3.bfm|~/runtime||||1
10946|tbase4.bfm|~/runtime||||1
10700|usatile1.bfm|~/runtime||||1
10701|usatile2.bfm|~/runtime||||1
10702|usatile3.bfm|~/runtime||||1
10703|usatile4.bfm|~/runtime||||1
10704|usatile5.bfm|~/runtime||||1
10705|usatile6.bfm|~/runtime||||1
10706|usatile7.bfm|~/runtime||||1
10707|usatile8.bfm|~/runtime||||1
10708|usatile9.bfm|~/runtime||||1
10709|usatile10.bfm|~/runtime||||1
10710|usatile11.bfm|~/runtime||||1
10711|usatile12.bfm|~/runtime||||1
#
# -----------------------------------------------------------------
# leap seconds (TAI-UTC) file
# -----------------------------------------------------------------
10301|leapsec.dat|~/lib/database/TD||||1
#
# -----------------------------------------------------------------
# polar motion and UT1-UTC file
# -----------------------------------------------------------------
10401|utcpole.dat|~/lib/database/CSC||||1
#
# -----------------------------------------------------------------
# earth model tags file
# -----------------------------------------------------------------
10402|earthfigure.dat|~/lib/database/CSC||||1
#
# -----------------------------------------------------------------
# directory where spacecraft ephemeris files are located
# NOTE: This line is used to specify a directory only!
#       The "file" field should not be altered.
# -----------------------------------------------------------------
10501|.|~/lib/database/EPH||||1
#
# -----------------------------------------------------------------
# JPL planetary ephemeris file (binary form)
# -----------------------------------------------------------------
10601|de200.eos|~/lib/database/CBP||||1
# -----------------------------------------------------------------
# End Toolkit support input files
# -----------------------------------------------------------------
#
#
?   SUPPORT OUTPUT FILES
# [ next line is for default location ]
! ~/supportoutput
#
#
# -----------------------------------------------------------------
# Toolkit support output files
# -----------------------------------------------------------------
#
# -----------------------------------------------------------------
# These files support the SMF log functionality. Each run will
# cause status information to be written to 1 or more of the Log
# files. To simulate DAAC operations, remove the 3 Logfiles
# between test runs.
```

```
# Remember: all executables within a PGE will contribute status
# data to the same batch of log files.
# -----------------------------------------------------------------
10100|LogStatus|~/runtime||||1
10101|LogReport|~/runtime||||1
10102|LogUser|~/runtime||||1
10103|TmpStatus|~/runtime||||1
10104|TmpReport|~/runtime||||1
10105|TmpUser|~/runtime||||1
10110|MailFile|~/runtime||||1
#
# -------------------------------------------------------------------------
# ASCII file which stores pointers to runtime SMF files in lieu of
# loading them to shared memory.
# -------------------------------------------------------------------------
10111|ShmMem|~/runtime||||1
#
# -----------------------------------------------------------------
# End Toolkit support output files
# -----------------------------------------------------------------
#
#
?   USER DEFINED RUNTIME PARAMETERS
#
# -----------------------------------------------------------------
# Pathfinder AVHRR/Land runtime parameters
# -----------------------------------------------------------------
601|requested_size_x|409
602|requested_size_y|128
603|wait_time|3
#
# -----------------------------------------------------------------
# These parameters are required to support the PGS_SMF_Send*
# tools. If the first parameter (TransmitFlag) is disabled, then
# none of the other parameters need to be set. By default, this
# functionality has been disabled. To enable, set TransmitFlag
# to 1 and supply the other 3 parameters with local information.
# -----------------------------------------------------------------
10109|TransmitFlag; 1=transmit,0=disable|1
10106|RemoteHost|fire@eos.hitc.com
10107|RemotePath|/fire2/toma/inbox
10108|EmailAddresses|toma@eos.hitc.com
#
# -------------------------------------------------------------------------
# Default location for processing host IP address.
# This is overridden by the environment variable PGS_HOST_PATH.
# -------------------------------------------------------------------------
10099|Local IP Address of 'ether'|155.157.31.87
#
#
?   INTERMEDIATE INPUT
# [ next line is for default location ]
! ~/runtime
#
#
?   INTERMEDIATE OUTPUT
# [ next line is for default location ]
! ~/runtime
#
#
?   TEMPORARY IO
# [ next line is for default location ]
! ~/runtime
#
# -----------------------------------------------------------------
# Pathfinder AVHRR/Land temporary file
# -----------------------------------------------------------------
901|test10.bin|||||
#
#
?   END
```