EED2-333-001

EOSDIS Evolution and Development-2 Contract

Release 9 SDP Toolkit Users Guide for the EOSDIS Evolution and Development-2 Contract

December 2017

Raytheon Company
Riverdale, Maryland

Release 9 SDP Toolkit Users Guide for the EED-2 Project

December 2017

Prepared Under Contract NNG105HZ39C
CDRL Item #23

RESPONSIBLE ENGINEER

E. Moghaddam Taaheri 12/21/17

Abe Taaheri, Toolkit Task Lead Date
EOSDIS Evolution and Development-2 (EED-2) Contract

SUBMITTED BY

Kristene Rodrigues 12/21/17

Kristene Rodrigues, Task 32 Lead Date
EOSDIS Evolution and Development-2 (EED-2) Contract

Raytheon Company
Riverdale, Maryland

EED2-333-001

This page intentionally left blank.

Preface

This document is a formal contract deliverable. It requires Government review and approval within 20 business days. Changes to this document will be made by document change notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
EED2 Contract
Raytheon Company
5700 Rivertech Court
Riverdale, MD 20737

This SDP Toolkit version 5.2.20 is directed at EOS instrument data providers who will deliver code to the ECS Release 9 DAACs. It is an engineering upgrade to Toolkit 5.2.19, delivered in March 2014. The user calling interface of the current version is the same as that of Toolkit 5.2.19.

The purpose of this document is to provide Earth Observing System (EOS) instrument data processing software developers and scientists with knowledge of Toolkit functionality: to provide a listing of routine calling sequences; and to provide detailed descriptions and examples of usage. The Toolkit will be used by developers at their Science Computing Facilities (SCFs) to develop EOS data production software and to prepare that software for integration into distributed active archive centers (DAACs). Subsequent usage of the Toolkit will be in conjunction with the services provided by the DAACs to produce, archive and distribute standard products. This document accompanies a software delivery that contains implementations of the tools described in the document. We note that this SCF version of the Toolkit contains provisions for error/status message, process control and file name handling by science software in lieu of an operational scheduling system. This handling will be via manual manipulation of UNIX files. This version also contains tools for creation and access of standard product metadata parameters as well as several added ancillary data files (e.g., a geoid model).

The hierarchical data format (HDF) has been selected by the Earth Observing System Data and Information System (EOSDIS) Project as the format of choice for standard product distribution. ECS has created the HDF-EOS extensions to HDF, which provide EOS specific HDF structures. For more information about HDF-EOS, see the HDF-EOS Library Users Guide. HDF is a *disk format* and *subroutine library* for storage of most kinds of scientific data. As a *disk format*, HDF files consist of a directory and an unordered set of binary data objects. Each directory entry describes the location, the type, and the size of these binary objects.

The *HDF subroutine library* is designed to be easy for C and FORTRAN programmers to use. The HDF library consists of callable routines, each of which belongs to a particular *interface*. Each interface within these layers address a particular HDF function or a particular HDF data

structure, such as arrays, tables, and annotations. Both HDF4 and HDF5 - based files are supported.

This Users Guide is accompanied by a Toolkit Primer. The Primer is intended to provide a concise explanation of individual tool usage, functionality and coding examples. The Primer will not contain details, appendices, requirements trace, and so on; that are contained in this Users Guide. The Primer is available at http://newsroom.gsfc.nasa.gov/sdptoolkit/primer/tkprimer.html

Other Toolkit related documents and links can be found at Toolkit web site:
http://newsroom.gsfc.nasa.gov/sdptoolkit/toolkit.html

The URL for the SDP Toolkit Frequently Asked Questions (FAQ) page is
http://newsroom.gsfc.nasa.gov/sdptoolkit/faq.html

You can also get there from the EDHS Home Page http://edhs1.gsfc.nasa.gov. Click on “ECS Development”, then Click on “Toolkit”. The "Toolkit Frequently Asked Questions (FAQ)" link is on the SDP Toolkit Page.

The technical point of contact within the EOS Data and Information System (EOSDIS) core System (ECS) project is:

Abe Taaheri: Abe_Taaheri@raytheon.com

An e–mail drop for user questions and comments is: RVL_PGSTLKIT@raytheon.com

Revision History

Document Number	Status/Issue	Publication Date	CCR Number
333-CD-605-001	Submitted as Final	May 2002	02-0419
333-CD-605-002	Submitted as Final	December 2002	02-1050
333-CD-605-003	Submitted as Final	April 2003	03-0234
333-EMD-001	Revision -	October 2003	03-0715
333-EMD-001	Revision 01	May 2004	04-0215
333-EMD-001	Revision 02	August 2004	04-0350
333-EMD-001	Revision 03	April 2005	05-0150
333-EMD-001	Revision 04	March 2006	06-0127
333-EMD-001	Revision 05	February 2008	07-0560
333-EEB-001	Revision -	July 2009	09-0255
333-EED-001	Revision -	August 2010	10-0298
333-EED-001	Revision - 01	January 2012	12-0003
333-EED-001	Revision - 02	March 2014	14-0052
EED2-333-001	Revision -	December 2017	

Abstract

The SDP Toolkit Users Guide describes Toolkit routine usage for science software developers, who will produce code to process instrument data. This document describes the overall design of the Toolkit, provides a general explanation of usage, and installation procedures on computer platforms for which software development and certification have been done. Detailed listings of routines, calling sequences, inputs and outputs and examples of usage are also provided. This current Users Guide is updated to match the Release 9 SDP Toolkit delivery.

Keywords*:* toolkit, metadata, HDF, HDF5, HDF-EOS, data, format, production, error, handling, process, control, geolocation, input, output, memory, management

This page intentionally left blank.

Contents

4. Toolkit Usage, Functionality, and Future Direction

5. Toolkit Installation and Maintenance

6. SDP Toolkit Specification

List of Figures

List of Tables

1. Introduction

1.1 Identification

The SCF Toolkit Users Guide (Contract Data Requirements List (CDRL) Item 023, Data Item Description (DID) EED-EDP-23) is a part of the Science Data Production (SDP) Toolkit delivery made under the Earth Observing System Data and Information System (EOSDIS) Evolution and Development-2 Contract (EED-2). It was first delivered in January 1994. The current Users Guide matches the Release 9 Toolkit delivery being made in December 2017. SCF Toolkit Users Guide for the ECS Project will be updated for each major release of the SDP Toolkit.

1.2 Scope

This Science Computing Facility (SCF) Toolkit version 5.2.20 is directed at EOS instrument data providers who will deliver code to the ECS Release 9 DAACs. It is an engineering update to Toolkit 5.2.19, delivered in March 2014. The user calling interface of the current version is the same as that of Toolkit 5.2.19. The SCF Toolkit Users Guide describes Toolkit routine usage for science software developers, who will produce code to process instrument data. The current version of the User's Guide is for the Release 9 Toolkit delivered code, however, the Toolkit will be updated as requirements are updated, certified and requirements for later platform instruments are determined. This document describes the overall design of the Toolkit, provides a general explanation of usage, and installation procedures on computer platforms for which software development and certification have been done. Detailed listings of routines, calling sequences, inputs and outputs and examples of usage are also provided.

1.3 Purpose and Objectives

This document is aimed at the EOS data production software developers and scientists who will use the SDP Toolkit to encapsulate their code in the distributed active archive center (DAAC) computing facilities. The purpose of the Toolkit is to provide an interface between instrument processing software and the production system environment. It sets up the context and environment to facilitate portability of code for the execution of production processes and the transfer of data sets and information to those processes. This interface will be implemented in the SCF development environment, along with additional utilities that will be used to emulate production environment services.

An important goal of the Toolkit is to facilitate the smooth transition and integration of code into the DAAC by abstracting out science process dependencies on external system architecture. Another goal is the provision of an interface into which application modules can be incorporated. This may include, for example, math packages; other specialized routines that can be commercial–off–the–shelf software (COTS); freeware; or user supplied modules. An effort will be made during development to incorporate and reuse existing application software modules.

This Users Guide will layout the high-level design of Toolkit and provide sufficient description of routines to show how EOS science software should incorporate the Toolkit interface.

In the description of the Toolkit routines, descriptive information is presented in the following format:

TOOL TITLE

NAME:	Procedure or routine name
SYNOPSIS:	
C:	C language call
FORTRAN:	FORTRAN77 or FORTRAN90 language call
DESCRIPTION:	Cursory description of routine usage
INPUTS:	List and description of data files and parameters input to the routine
OUTPUTS:	List and description of data files and parameters output from the routine
RETURNS:	List of returned parameters indicating success, failure, etc.
EXAMPLES:	Example usage of routine
NOTES:	Detailed information about usage and assumptions
REQUIREMENTS:	Requirements from *PGS Toolkit Specification*, Oct. 93 which the routine satisfies

1.4 Status and Schedule

This Users Guide accompanies a set of toolkit routines, delivered in December 2017. Table 1–2 below gives a complete listing; brief description; and delivery dates of Toolkit software available to users. We note also several important related schedule items:

- April 1995—IDL was selected as the Toolkit graphics package of choice.
- July 1995—Release Toolkit A delivery, including prototype HDF-EOS swath structure software
- July 1995—Delivery (to the EOS community) of a draft HDF-EOS standard and users guide.
- January 1996—ECS Interim Release 1 (Ir1)
- May 1996— Release A SCF Toolkit delivery.
- July 1996 HDF-EOS version 1.0 delivery
- November 1996 updated HDF-EOS and SCF Toolkit delivery
- April 1997 Release B.0 SCF Toolkit and HDF-EOS 2.0 delivery
- October 1997 Version 2.0 SDP Toolkit and HDF-EOS 2.1 delivery
- March 1998 Version 2.0 SDP Toolkit and HDF-EOS 2.2 delivery
- October 1998 Version 2.0 SDP Toolkit and HDF-EOS 2.3 delivery

- January 1999 Version 2.0 SDP Toolkit and HDF-EOS 2.4 delivery
- June 1999 Version 2.0 SDP Toolkit and HDF-EOS 2.5 delivery
- February 2000 Release 5B SDP Toolkit and HDF-EOS 2.6 delivery
- November 2000 Release 5B SDP Toolkit and HDF-EOS 2.7 delivery
- November 2002 Release 6A SDP Toolkit and HDF-EOS 2.8 and HDF-EOS5.1.3 delivery
- April 2003 Release 6A SDP Toolkit and HDF-EOS 2.9 and HDF-EOS5.1.5 delivery
- October 2003 Release 6A SDP Toolkit, HDF-EOS 2.10 and HDF-EOS5.1.6 delivery
- May 2004 Release 7 SDP Toolkit, HDF-EOS 2.11 and HDF-EOS5.1.7 delivery
- August 2004 Release 7 SDP Toolkit, HDF-EOS 2.12 and HDF-EOS5.1.8 delivery
- April 2005 Release 7 SDP Toolkit, HDF-EOS 2.13 and HDF-EOS5.1.9 delivery
- March 2006 Release 7 SDP Toolkit, HDF-EOS 2.14 and HDF-EOS5.1.10 delivery
- February 2008 Release 7 SDP Toolkit, HDF-EOS 2.15 and HDF-EOS5.1.11 delivery
- July 2009 Release 7 SDP Toolkit, HDF-EOS 2.16 and HDF-EOS5.1.12 delivery
- August 2010 Release 7 SDP Toolkit, HDF-EOS 2.17 and HDF-EOS5.1.13 delivery
- January 2012 Release 8 SDP Toolkit, HDF-EOS 2.18 and HDF-EOS5.1.14 delivery
- March 2014 Release 8 SDP Toolkit, HDF-EOS 2.19 and HDF-EOS5.1.15 delivery
- December 2017 Release 9 SDP Toolkit, HDF-EOS 2.20 and HDF-EOS5.1.16 delivery

Table 1–1 provides a key to the tool names and the section where the specific tools can be located.

Table 1-1. Toolkit Routine Key

Key	Class	Section
AA	Ancillary Data Access	6.3.2
CBP	Celestial Body Position	6.3.3
CSC	Coordinate System Conversion	6.3.4
CUC	Constant and Unit Conversions	6.3.7
DEM	Digital Elevation Model access	6.3.1
EPH	Ephemeris Data Access	6.2.6
GCT	Geo Coordinate Transformation	6.3.5
IO	Input Output (File I/O)	6.2.1
MEM	Memory Management	6.2.4
MET	Metadata Access	6.2.1
PC	Process Control	6.2.3
SMF	Status Message File (Error/Status)	6.2.2
TD	Time Date Conversion	6.2.7
XML	Internal XML conversion of ODL Metadata	

In Table 1–2 a list of Toolkit routines is given, with delivery data and page number references in this Users Guide. Table 1–2 lists Toolkit routines alphabetically by class as defined in the key below. The class keyword follows the Product Generation System (PGS) keyword (i.e., PGS_AA).

Table 1-2. Toolkit Routine Listing (1 of 7)

Tool Name	Description	Date	Page
Pccheck	Use to verify that a process control file (PCF) is syntactically correct	10-94 7-95	6-187
PGS_AA_2Dgeo	Allows access to 2 dimensional data sets, e.g., sea–ice	10-94, 2-95, 7-95, 4-96	6-355
PGS_AA_2Dread	Allows access to 2 dimensional data sets, e.g., sea–ice	10-94, 2-95, 4-96	6-365
PGS_AA_3Dgeo	Allows access to 3 dimensional data sets,e.g., atmospheric humidity	10-94 2-95 4-96	6-360
PGS_AA_3Dread	Allows access to 3 dimensional data sets,e.g., atmospheric model	10-94 2-95 4-96	6-370
PGS_AA_dcw	Returns the surface types (land, sea, coast), and nation–state to be determined (TBD) for a user defined set of locations	10-94 4-96	6-339
PGS_AA_dem	Locates heights from specified digital elevation model (DEM) corresponding to each of the locations specified	2-95, 7-95 4-96	6-342
PGS_AA_PeVA_integer	Searches in a specified file for the parameter and returns the value of that parameter which is an integer	10-94 2-95, 7-95 4-96	6-353
PGS_AA_PeVA_real	Searches in a specified file for the parameter and returns the value of that parameter which is a real(float)	10-94 2-95, 7-95 4-96	6-350
PGS_AA_PeVA_string	Searches in a specified file for the parameter and returns the value of that parameter which is a text string	10-94 2-95, 7-95 4-96	6-347
PGS_CBP_body_inFOV	Given instrument parameters, returns a flag to indicate whether any of the user–selected major celestial bodies (sun, moon, etc.) are in the instrument field–of–view.	2-95, 7-95	6-390
PGS_CBP_Earth_CB_Vector	Computes the Earth centered inertial (ECI) frame vector from the Earth to the sun, moon, or planets at a given time, or range of time(s)	4-94, 10-94 7-95	6-378
PGS_CBP_Sat_CB_Vector	Computes the ECI vector from the spacecraft to the sun, moon, or planets at a given time or range of time(s)	4-94, 10-94 7-95	6-382
PGS_CBP_SolarTimeCoords	Computes local solar time, and right ascension and declination of the sun, for a given standard time and position on the surface of the Earth	4-94, 10-94 7-95	6-386
PGS_CSC_DayNight	Determines whether a given point on the Earth is in day, night or twilight, at a given time	10-94 7-95	6-491
PGS_CSC_Earthpt_FixedFOV	For a fixed field of view obtains the Coordinated Universal Time (UTC) time interval and the starting time that an Earth point is within the field–of–view, within a specified time window	4-96	6-451

Table 1-2. Toolkit Routine Listing (2 of 7)

Tool Name	Description	Date	Page
PGS_CSC_Earthpt_FOV	For a field of view defined by a table of coordinates (accessed externally), and a known motion of the boresight vector as a function of time, obtains the Coordinated Universal Time (UTC) time interval and the starting time that an Earth point is within the field–of–view, within a specified time window	2-95, 7-95	6-457
PGS_CSC_ECItoECR	Transforms a vector from the ECI frame to the ECR frame.	10-94 7-95	6-407
PGS_CSC_ECItoORB	Transforms a vector in the ECI Coordinate system to a vector in the Orbital Coordinate System	7-95	6-437
PGS_CSC_ECItoSC	Transforms a vector in the ECI coordinate system to the Spacecraft Coordinate System.	10-94	6-421
PGS_CSC_ECRtoECI	Transforms a vector from the ECR system to the ECI system.	10-94 7-95	6-411
PGS_CSC_ECRtoGEO	Transforms a vector from rectangular ECR coordinates to geodetic coordinates.	10-94 7-95	6-415
PGS_CSC_GEOtoECR	Transforms a vector from geodetic coordinates to ECR coordinates.	10-94 7-95	6-418
PGS_CSC_GetFOV_Pixel	Computes the projection of (geolocates) a pixel.	4-94, 10-94 2-95, 7-95	6-469
PGS_CSC_GrazingRay	For rays that miss Earth limb, this function finds the nearest miss point on the ray and corresponding surface point. For rays that strike the Earth, it outputs instead the coordinates of the midpoint of the chord of the ray within the ellipsoid and surface coordinates of the intersection nearest the observer	4-97	6-510
PGS_CSC_GreenwichHour	Returns the Greenwich Hour Angle of the vernal equinox, which is equal to Greenwich sidereal time, in the ECI frame, at a given time.	10-94	6-499
PGS_CSC_J2000toTOD	Transform from ECI J2000 to ECI True of Date	4-96	6-485
PGS_CSC_nutate2000	Transforms 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	7-95 4-96	6-481
PGS_CSC_ORBtoECI	Transforms vector in orbital coordinate system to vector in ECI coordinate system	7-95	6-441
PGS_CSC_ORBtoSC	Transforms a vector from orbital to spacecraft coordinates.	10-94 7-95	6-433
PGS_CSC_precs2000	Precesses 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)	7-95	6-477
PGS_CSC_SCtoECI	Transforms a vector from spacecraft to ECI coordinates.	10-94	6-425
PGS_CSC_SCtoORB	Transforms a vector from spacecraft to orbital coordinates.	10-94 7-95	6-429
PGS_CSC_SpaceRefract	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	7-95 4-96	6-464
PGS_CSC_SubSatPoint	Returns the position and velocity vector of the sub–satellite point or nadir of the satellite on the Earth's surface. Also returns the rate of change of altitude off the ellipsoid.	4-94, 10-94	6-445
PGS_CSC_TODtoJ2000	Transform from ECI True of Date to ECI J2000 Coordinates	4-96	6-488
PGS_CSC_wahr2	Calculates nutation angles	7-95	6-496
PGS_CSC_ZenithAzimuth	Returns zenith and azimuth angles of viewing vector or a celestial body	10-94 2-95	6-503
PGS_CUC_Cons	Accesses constant values from a predetermined input file	2-95	6-529

Table 1-2. Toolkit Routine Listing (3 of 7)

Tool Name	Description	Date	Page
PGS_CUC_Conv	Accesses conversion slope and intercept values, needed to convert between units	2-95	6-531
PGS_DEM_Close	Close a DEM dataset	4-97	6-297
PGS_DEM_DataPresent	Check for Valid DEM Data Point	4-97	6-300
PGS_DEM_GetMetadata	Extract Metadata from the DEM	4-97	6-323
PGS_DEM_GetPoint	Return Data at Specified DEM Point	4-97	6-309
PGS_DEM_GetQualityData	ACCESS DEM Quality Data	4-97	6-328
PGS_DEM_GetRegion	Return Data from a Specified Region of the DEM	4-97	6-316
PGS_DEM_GetSize	Return Size of Specified DEM Region	4-97	6-334
PGS_DEM_Open	Open a DEM dataset	4-97	6-294
PGS_DEM_SortModels	Check for Data in a Specified Region of the DEM	4-97	6-304
PGS_EPH_EphemAttit	Provides access to spacecraft ephemeris and attitude data for a given time range, interpolates the state vectors and spacecraft attitude to a specified time. Retains quality flags	4-94, 10-94 2-95, 7-95 4-96	6-211
PGS_EPH_GetEphMet	gets metadata associated with toolkit spacecraft ephemeris files	11-96	6-223
PGS_EPH_ManageMasks	get and/or set the values of the ephemeris and attitude quality flags masks		6-228
PGS_EPH_Eph_Att_unInterpolate	Gets actual (without interpolation) ephemeris and/or attitude records for the specified spacecraft if the number of records for ephemeris is the same as that of the attitude for the requested time period	9-02	6-217
PGS_EPH_UnInterpEphAtt	Gets actual (without interpolation) ephemeris and/or attitude records for the specified spacecraft even if the number of records for ephemeris is not the same as that of attitude for the requested time period	10-03	6-217
PGS_GCT_Init	Performs Geo–coordinate transformation initialization for the given projection with the given parameters	2-95, 7-95	6-520
PGS_GCT_Proj	Performs Geo–coordinate transformations for the given projection in the forward and inverse directions	2-95, 7-95	6-523
PGS_IO_Gen_Close	Close non–HDF file	4-94, 10-94	6-46
PGS_IO_Gen_CloseF	Close non–HDF file FORTRAN	10-94 7-95	6-48
PGS_IO_Gen_Open	Open non–HDF file	4-94, 10-94 7-95	6-39
PGS_IO_Gen_OpenF	Open non–HDF file FORTRAN 77	10-94 2-95, 7-95	6-42
PGS_IO_Gen_Temp_Delete	Permanently delete a temporary file	4-94, 10-94 2-95, 7-95	6-95
PGS_IO_Gen_Temp_Open	Open temporary file	4-94, 10-94 2-95	6-86
PGS_IO_Gen_Temp_OpenF	Open temporary file FORTRAN 77 & 90	10-94 2-95	6-91
PGS_IO_L0_Close	Closes a virtual data set that was opened with a call to PGS_IO_L0_Open.	2-95 4-96 2-00	6-27

Table 1-2. Toolkit Routine Listing (4 of 7)

Tool Name	Description	Date	Page
PGS_IO_L0_File_Sim	Creates a file of simulated Level 0 data	2-95 4-96 2-00	6-29
PGS_IO_L0_GetHeader	Gets the header and footer data for the currently open physical file	2-95 4-96 2-00	6-17
PGS_IO_L0_GetPacket	Gets a single packet from the specified Level 0 Virtual Data Set	2-95 4-96 2-00	6-22
PGS_IO_L0_Open	Open a Virtual Level 0 Data Set	2-95 4-96 2-00	6-6
PGS_IO_L0_SetStart	Sets the specified open virtual data set so that the next call to PGS_IO_L0_GetPacket will read the first packet at or after the specified time	2-95 4-96 2-00	6-11
PGS_IO_L0_SetStartCntPkts	Sets the specified open virtual data set so that the next call to PGS_IO_L0_GetPacket will read the first packet at or after the specified time and tracks the number of packets skipped in the current file.	4-97 2-00	6-14
PGS_MEM_Calloc	Allocates an array of arbitrarily sized elements, initializing them to zero, in memory	10-94 7-95 2-00	6-536
PGS_MEM_Free	Deallocates memory that was previously allocated	10-94 7-95	6-541
PGS_MEM_FreeAll	Deallocates all memory that was previously allocated within a process	10-94 7-95	6-542
PGS_MEM_Malloc	Allocates an arbitrary number of bytes in memory	10-94 7-95	6-534
PGS_MEM_Realloc	Reallocates the number of bytes requested	10-94 7-95	6-538
PGS_MEM_ShmAttach	Used by an executable to attach to an existing shared memory segment	10-94	6-197
PGS_MEM_ShmCreate	Used to create a shared memory segment	10-94	6-195
PGS_MEM_ShmDetach	Used to detach a shared memory segment from a process that attached it	10-94	6-199
PGS_MEM_ShmRead	FORTRAN Read from Shared Memory	4-96	6-201
PGS_MEM_ShmWrite	FORTRAN Write to Shared Memory	4-96	6-203
PGS_MEM_Zero	Initializes a memory block or structure to zero	10-94 7-95	6-540
PGS_MET_GetConfigData	Enables the user to get the values of Config data parameters held in the PC table	7-95 4-96	6-72
PGS_MET_GetPCAttr	Retrieves parameter values from the PC table which are either located as HDF attributes on product files or in separate ASCII files	7-95 4-96	6-67
PGS_MET_GetSetAttr	Enables the user to get the values of metadata parameters which are already set by the initialization procedure	7-95 4-96	6-64
PGS_MET_Init	Initializes a metadata configuration file (MCF)	7-95 4-96	6-52
PGS_MET_Remove	Contains PGS_MET_Remove() which frees the memory held by the metadata configuration file (MCF) and data dictionary object description language (ODL) representations	7-95 4-96	6-80
PGS_MET_SetAttr	Enables the user to set the value of metadata parameters	7-95 4-96	6-56

Table 1-2. Toolkit Routine Listing (5 of 7)

Tool Name	Description	Date	Page
PGS_MET_SetMultiAttr	Enables the user to set the value of multi value metadata parameters and modify NUM_VAL value to correct value	3-02	6-61
PGS_MET_SDstart	Enables opening and obtaining SD ID for HDF files of HDF4 and HDF5 type	3-02	6-81
PGS_MET_SDend	Enables closing HDF files of HDF4 and HDF5 type that were opened by a call to PGS_MET_Sdstart	3-02	6-83
PGS_MET_Write	Enables the user to write different groups of metadata to separate HDF attributes	7-95 4-96	6-75
PGS_PC_GenUniqueID	Used to generate a unique product identifier. May be attached to file metadata to facilitate tracking of production output	10-94 4-96	6-173
PGS_PC_GetConfigData	May be used to access run–time parameters in the PGE	10-94 4-96	6-175
PGS_PC_GetConfigDataCom	May be used to access run–time parameters at the shell level	2-95 4-96	6-152
PGS_PC_GetFileAttr	Used to retrieve the attribute string that contains the metadata for a Product file	10-94 4-96	6-181
PGS_PC_GetFileAttrCom	Used at the shell level to retrieve an attribute "stream" that contains the metadata for a Product file	2-95 4-96	6-154
PGS_PC_GetFileByAttr	Used to retrieve the specific instance of a product file that satisfies the search criteria, defined by a user–supplied method, applied to the metadata of each product file instance	10-94 4-96	6-184
PGS_PC_GetFileSize	Get the size of a file in the PCF.	4-97	6-192
PGS_PC_GetFileSizeCom	Get the size of a file in the PCF at the shell level.	4-97	6-161
PGS_PC_GetNumberOfFiles	May be used to query the number of file instances that are associated with a particular product file	10-94 4-96	6-178
PGS_PC_GetNumberOfFilesCom	May be used, at the shell level, to query the number of file instances that are associated with a particular product file	2-95 4-96	6-153
PGS_PC_GetReference	Used to obtain a physical file pathname from a logical identifier for a particular product file	10-94 4-96	6-166
PGS_PC_GetReferenceCom	Used at the shell level to obtain a physical file pathname from a logical identifier for a particular product file	2-95 4-96	6-149
PGS_PC_GetReferenceType	Tool may be used to ascertain the type of file reference which is associated with a logical identifier within the science software	7-95 4-96	6-169
PGS_PC_GetTempReferenceCom	Used at the shell level to obtain a physical file pathname from a logical identifier for a particular temporary, or intermediate file	2-95, 7-95 4-96	6-157
PGS_PC_GetUniversalRef	Used to obtain a universal reference from a logical identifier	4-96	6-189
PGS_PC_InitCom	Used, prior to PGE execution, to establish a working environment for the SDP Toolkit	2-95 7-95 4-96	6-148
PGS_PC_Shell.sh	Provides an integrated environment for the SDP Toolkit and a PGE	2-95, 7-95 4-96, 11-96 10-97	6-145
PGS_PC_TempDeleteCom	Used at the shell level to delete the temporary file currently associated with a particular logical identifier	2-95 4-96	6-160
PGS_PC_TermCom	Used, following PGE termination, to cleanup the resources used by the SDP Toolkit	2-95 4-96	6-163
PGS_SMF_Begin	Signal SMF that function has started	4-96	6-137

Table 1-2. Toolkit Routine Listing (6 of 7)

Tool Name	Description	Date	Page
PGS_SMF_CreateMsgTag	May be used to generate a unique message identifier	10-94 4-96	6-116
PGS_SMF_End	Signal SMF that function has ended	4-96	6-138
PGS_SMF_GenerateStatusReport	Used to add user–defined status reports to the Status Report Log file	10-94 4-96	6-120
PGS_SMF_GetActionByCode	Provide the means to retrieve an action string associated with a specific mnemonic code	10-94 4-96	6-114
PGS_SMF_GetInstrName	Used to retrieve the instrument name from a given error/status code	4-94, 10-94 4-96	6-118
PGS_SMF_GetMsg	Provide the means to retrieve a previously set message from the static buffer PGS_SMF_Set....	4-94, 10-94 4-96	6-113
PGS_SMF_GetMsgByCode	Provide the means to retrieve the message string corresponding to a specific mnemonic code	10-94 4-96	6-112
PGS_SMF_GetToolkitVersion	This function returns a string describing the current version of the Toolkit.	4-97	6-103
PGS_SMF_SendRuntimeData	Provide a means for the user to transmit a package of runtime data to the SCF in the event of an unhandled system exception	10-94 2-95 4-96	6-122
PGS_SMF_SetArithmeticTrap	Used to specify a signal handling function to perform in the event that an error arithmetic operation has occurred.	TBD	6-139
PGS_SMF_SetDynamicMsg	Provide the means to set a user–defined error/status message in response to the outcome of some segment of processing.	10-94 4-96	6-109
PGS_SMF_SetStaticMsg	Provide the means to set a predefined error/status message in response to the outcome of some segment of processing.	4-94, 10-94 4-96	6-107
PGS_SMF_SetUNIXMsg	Provides the means to retain UNIX error messages for later retrieval	4-94, 10-94 4-96	6-104
PGS_SMF_TestErrorLevel	Will return a Boolean value indicating whether or not the returned code has status level 'E'	10-94 4-96	6-126
PGS_SMF_TestFatalLevel	Will return a Boolean value indicating whether or not the returned code has status level 'F'	10-94 4-96	6-128
PGS_SMF_TestMessageLevel	Will return a Boolean value indicating whether or not the returned code has status level 'M'	10-94 4-96	6-129
PGS_SMF_TestNoticeLevel	Will return a Boolean value indicating whether or not the returned code has status level 'N'	10-94 4-96	6-133
PGS_SMF_TestStatusLevel	Will return a defined status level constant	4–94, 10-94 4-96	6-134
PGS_SMF_TestSuccessLevel	Will return a Boolean value indicating whether or not the returned code has status level 'S'	10-94 4-96	6-132
PGS_SMF_TestUserInfoLevel	Will return a Boolean value indicating whether or not the returned code has status level 'U'	10-94 4-96	6-131
PGS_SMF_TestWarningLevel	Will return a Boolean value indicating whether or not the returned code has status level 'W'	10-94 4-96	6-130
PGS_TD_ASCIItime_AtoB	Converts binary time values to ASCII Code B time values of the form year_month_day_time_of_day in the Consultative Committee on space Data Systems (CCSDS) format	10-94	6-257
PGS_TD_ASCIItime_BtoA	Converts binary time values to ASCII Code A time values of the form year_month_day_time_of_day in the CCSDS format	10-94	6-259

Table 1-2. Toolkit Routine Listing (7 of 7)

Tool Name	Description	Date	Page
PGS_TD_GPStoUTC	Converts to Coordinated Universal Time (UTC) time value from Global Positioning System (GPS) time by converting to internal time, adding the GPS_minus_UTC_leapseconds from the leapseconds file, and converting to GPS format following CCSDS ASCII standard A	10-94 7-95	6-263
PGS_TD_LeapSec	Find leap second value	4-96	6-282
PGS_TD_SCtime_to_UTC	Converts spacecraft clock time to UTC for EOS platforms or for foreign spacecraft	4-94, 10-94, 2-00	6-254
PGS_TD_TAItoGAST	Converts International Atomic Time (TAI) (toolkit internal time) to Greenwich apparent sidereal time (GAST) expressed as the hour angle of the true vernal equinox of date at the Greenwich meridian (in radians)	7-95	6-249
PGS_TD_TAIjdtoTAI	Converts TAI Julian date to time in TAI seconds since 12 AM UTC 1-1-1993.	4-96	6-247
PGS_TD_TAItoTAIjd	Converts time in TAI seconds since 12 AM UTC 1-1-1993 toTAI Julian date.	4-96	6-245
PGS_TD_TAItoUTC	Converts a toolkit TAI time value to UTC time	4–94, 10-94	6-243
PGS_TD_TimeInterval	Computes the elapsed TAI time in seconds between any two epochs after January 1, 1958	10-94	6-271
PGS_TD_UTCtoGPS	Converts UTC time value to GPS time by converting to internal time, adding the GPS_minus_UTC_leapseconds from the leapseconds file, and converting to GPS format following CCSDS ASCII standard A	10-94 7-95	6-261
PGS_TD_UTCtoTAI	Converts UTC time to TAI time by first converting UTC to internal time and then adding the TAI_minus_UTC_leapseconds from the leapseconds file	4-94, 10-94	6-240
PGS_TD_UTCtoTDBjed	UTC to Barycentric Dynamical Time (TDB) time conversion	10-94	6-268
PGS_TD_UTCtoTDTjed	UTC to Terrestrial Dynamical Time (TDT) time conversion	10-94	6-265
PGS_TD_UTCtoUT1	Converts UTC to UT1 time	10-94	6-277
PGS_TD_UTCtoUT1jd	Converts UTC time in CCSDS ASCII Time Code to UT1 time as a Julian date	7-95	6-280
PGS_TD_UTCjdtoUTC	Converts UTC as a Julian date to UTC in CCSDS ASCII Time Code A format.	4-96	6-275
PGS_TD_UTCtoUTCjd	Converts UTC in CCSDS ASCII Time Code A format to UTC as a Julian date.	4-96	6-273
PGS_TD_UTC_to_Sctime	Converts UTC to Spacecraft clock time for EOS standard of Foreign Spacecraft	10-94 2-00	6-251
Smfcompile	Provides means to store messages in files that are accessed at run time to get the message text.	4-94, 10-94 2-95	6-141

Note for Table 1-2: If more than one date is in the delivery column this indicates a re–delivery of that tool.

Table 1-3. Tool Changes for Release 9 Toolkit Delivery

Tool Name	Type of Change
INSTALL-Toolkit	updated to reflect corrections from bugs
Toolkit	updated for more current compilers
Toolkit	Freeware packages updated to current versions
Toolkit	all user support bugs fixed

1.5 Document Organization

The document is organized as follows:

Section 1 Introduction—Presents the scope and purpose of this document.

Section 2 Related Documentation—Provides a bibliography of reference documents for the science data production (SDP) Toolkits organized by parent and applicable documents.

Section 3 Toolkit Design Overview—Provides the philosophy and high-level description of the Toolkit

Section 4 Toolkit Usage and Functionality—Describes the functionality to be provided in the SCF and follow–on SDP versions of the Toolkit.

Section 5 Toolkit Installation—Contains installation procedures for the machines for which Version 1 of the Toolkit has been certified.

Section 6 SDP Toolkit Specification—Contains calling sequences, description and usage examples for Toolkit routines.

Appendix A Assumptions

Appendix B Status Message File (SMF) Creation and Usage Guidelines

Appendix C Defining Process Control Files

Appendix D Ancillary Data Access Tools

Appendix E Example of Usage of Level 0 Access Tools

Appendix F Level 0 File Formats

Appendix G PGS_GCT Information Relating To Interface Specification

Appendix H PGS_CUC_Cons—Example Standard Constants File

Appendix I PGS_CUC_Conv—Input File Provided with the UdUnits Software

Appendix J Population of Granule Level Metadata using the SDP metadata tools

Appendix K POSIX Systems Calls Usage

Appendix L Ephemeris and Attitude File Formats

2. Related Documentation

2.1 Parent Documents

The parent documents are the documents from which this SDP Toolkit Users Guide's scope and content are derived.

	EED Task 02 Statement of Work for Providing ECS/ECHO Sustaining Engineering and Continuous Evolution
423-CDRD-002	Contract Data Requirements Document for EED Task 01, 02, 03
423-46-01	Functional and Performance Requirements Specification for Earth Observing System Data and Information System (EOSDIS) Core System Science System
none	Goddard Space Flight Center, The PGS Toolkit Study Report, Version 1.9

2.2 Applicable Documents

The following documents are referenced within this SDP Toolkit Users Guide, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

EED2-170-001	Release 9 HDF-EOS Library User's Guide for the ECS Project, Volume 1: Overview and Examples
EED2-170-002	Release 9 HDF-EOS Library User's Guide for the ECS Project, Volume 2: Function Reference Guide
EED2-175-001	Release 9 HDF-EOS5 Library User's Guide for the ECS Project, Volume 1: Overview and Examples
EED2-175-002	Release 9 HDF-EOS5 Library User's Guide for the ECS Project, Volume 2: Function Reference Guide
445-TP-002	Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project, Technical Paper
194-WP-924	Level 0 Data Issues for the ECS Project, White Paper
GSFC 50-003-04	Goddard Space Flight Center, EOSDIS Version 0 Data Product Implementation Guidelines (V1.0), 3/1/94

CCSDS 301.0-B-2	Consultative Committee for Space Data Systems (CCSDS) Recommendation for Space Data System Standards: Time Code Formats, Issue 2, 4/90
IEEE Std 1003.1	Institute of Electrical and Electronics Engineers; POSIX Part 1: System Application Program Interface (API)[C Language]
IEEE Std 1003.9	Institute of Electrical and Electronics Engineers; POSIX FORTRAN77 Language Interfaces, Part 1: Binding for System Application Program Interface [API]
none	Computer Science Corporation; Upper Atmosphere Research Satellite (UARS) Lessons Learned for EOS: Report 1—Design and Implementation (ending December 21, 1993); 5/92
none	University of Illinois/National Center for Supercomputing Applications; NCSA HDF Calling Interfaces and Utilities, Version 3.2; 3/93
none	University of Illinois; Getting Started With HDF, 1993
none	Wertz, J.R., Spacecraft Attitude Determination and Control, Reidel Publishing Co., 1984.

2.3 Information Documents

The following Internet link to a document/information, although not directly applicable, amplifies or clarifies the information presented in this document. This reference is not binding on this document.

Please note that Internet links cannot be guaranteed for accuracy or currency

194-815-SI4	SDP Toolkit Primer (current version available through http://newsroom.gsfc.nasa.gov/sdptoolkit/primer/tkprimer.html)

3. Toolkit Design Goals

The PGS Toolkit Requirements Specification served to create a specification for a compendium of tools that meet both ECS system requirements and the needs of the EOS science instrument data producers. The SDP Toolkit User's Guide represents the culmination of efforts to design tools that satisfy those criteria. In order to create that design, several broad features were devised to give the Toolkit a sense of continuity such that it may be considered a single tool with far-reaching capabilities.

3.1 Foundations

In order to ensure a high degree of portability and maintainability across a wide variety of computer platforms, the SDP Toolkit has been designed to conform to the POSIX.1 standard. With a few exceptions, this goal is met in the current implementation. Cases where a vendors, operating system or compiler implementation prevents strict adherence to the Portable Operating System Interface for Computer Environments (POSIX) standard will be minimized and worked as the standard matures. Additionally, some components of the Toolkit have been designed to incorporate proven COTS and other heritage software to provide functionality that is largely accepted by the user community and can be easily integrated into the Toolkit.

3.2 Nomenclature

The naming of the tools has been standardized to include two prefixes: one to denote its membership in the family of SDP tools and the other to indicate the general area of functionality covered by the tool. For example, a Toolkit routine that performs a time conversion will be prefixed with 'PGS_TD_'. The remaining portion of each name will be detailed enough to indicate the explicit functionality performed by the tool (e.g., "PGS_TD_UTCtoTAI").

3.3 Consistency

This feature was achieved by the creation of a method for setting and retrieving status values and status messages through the use of pre–defined error and status return codes and associated Toolkit routines. Some of these return codes are defined by the SDP system, but most of them will be defined by the users themselves to give them maximum control over their processing. All the SDP Toolkit routines have been designed to adhere to this status return mechanism; likewise, all the user developed software should incorporate this mechanism as well. The widespread use of this feature will serve to create software that is consistent in its approach to error handling and status reporting, is more readable, adheres to principles of modularity, and is easier to maintain.

3.4 Hierarchical Design

Finally, the SDP Toolkit was designed to provide different levels of service, depending on the requirements of the developer. Primarily, the Toolkit was designed to provide for all the necessary system–level interfaces. However, much of the Toolkit functionality incorporates value–added features to provide a higher level of service for the developers creating higher–order algorithms. In order to accomplish this, many of the Toolkit routines are designed to use the services of lower–level Toolkit routines. Some of the tools, such as the memory management routines are only required to have one or two levels of service; whereas others, like the ancillary data I/O routines, may have several different levels of service. It is important to note that whatever level of service is required, the Toolkit routine that provides that service will have been designed to use the services of a lower level Toolkit routine. This means that the applications programmer can use any of the Toolkit routines to develop their own level of service if there is not an explicit Toolkit routine that provides it.

3.5 Units

Generally, in the CBP, CSC, TD, and EPH sections of the Toolkit all physical quantities are in Standard International (SI) units, and all angles are in radians. The only exceptions to the use of SI units are a few cases where a "time" such as a Greenwich "time" that is really a measure of Earth rotation may be given in radians, or (for Julian Date) days, instead of seconds - please consult the individual tool entries on this issue. In some of the AA and GCT tools specialized units appropriate to the relevant data set may be used; please consult the individual entries.

The HDF subsetting functions use SI seconds.

Users who wish to work in units other than those in the Tools are urged to use great caution. For example, the tools that transform between the spacecraft reference frame and Earth-centered reference frames take into account the displacement of the spacecraft (in meters) from Earth center, when the user supplies other than a unit vector. (For unit vector input. only the direction is transformed). To use these transformations on vectors denominated, for example, in kilometers would result in nonsense.

3.6 Ranges and Limits of Validity; unit vectors

The following material applies to the CBP, CSC, TD, and EPH tools; the AA and GCT tools may follow different rules which are explained in the appropriate sections.

On output, all angles that represent a longitude or azimuth will be in the range $(-\pi, \pi)$, but on input the Toolkit is more forgiving: no limit is imposed, although most library trigonometric functions tend to lose accuracy when the argument is very large. By keeping the input range open this way we hope to simplify the task of the user who may, for example, want to transform from geodetic coordinates to rectangular coordinates a patch of the Earth's surface that bridges the longitude discontinuity at or near the international date line. There is no harm in entering values larger than π or less than $-\pi$ as derived, say, from offsets. Latitude is in the range $(-\pi/2, \pi/2)$. Nadir and Zenith angles are in the range $(0, \pi)$. Altitude can be arbitrary, but some

tools return warnings or balk, with an error return, if a questionable altitude is detected; see the individual descriptions. Referring to Section 3.5, here again is a case where the inadvertent input of coordinates in kilometers (which the tools would take to be meters) could result in worthless output and a warning message, only, that the spacecraft was "subterranean."

In many cases, CSC group tools require a unit vector input. The varying accuracies of different platforms, and the danger of algorithmic error in case of inputting a non-unit vector where a unit vector is called for, dictated that the Toolkit simply make a normalized copy of the vector for internal use anyway. Thus, users need not, in practice, normalize "unit vectors" supplied to our CSC functions. On output, when a unit vector is promised, however, a unit vector will be produced.

Certain time streams have limited range by the nature of their definition, as explained in the TD section. Generally, the broadest range of times is encompassed by the Julian Date time streams, but Toolkit time, secTAI93, will yield microsecond precision from 1960 to 2135 AD on 32 bit platforms.

The algorithms have been carefully chosen to preserve machine word precision where possible, but a few transformations are subject to some limitations explained in the individual entries. For example, as noted by Galileo and Copernicus, the apparent velocity of the Sun or a planet as viewed in a reference frame rotating with the Earth is absurdly large; therefore we do not calculate such velocities past the mean distance to the Moon.

3.7 Aging and Maturation Effects

Any tools, such as geolocation functions, that depend on a precise knowledge of Earth rotation, yield answers that depend ultimately on measurement; Earth rotation cannot be predicted well enough to allow ultra-precise real time geolocation! Therefore, along with leap seconds data, the Toolkit imports, weekly, data files on Earth rotation from the U.S. Naval Observatory. Users who want precise Earth position can get it within a few centimeters, but they have to wait a week till the latest file is in! Users content with meter accuracy can process in real time, but if they reprocess later, their geolocation answers may change by several centimeters, or even a meter. For more details, see the SDP Toolkit FAQ at http://newsroom.gsfc.nasa.gov/sdptoolkit/faq.html.

This page intentionally left blank.

4. Toolkit Usage, Functionality, and Future Direction

4.1 Introduction

This User's Guide addresses the usage of the SCF version of the SDP Toolkit. The purpose of the SCF development environment and Toolkit is (1) to provide development Toolkit functions that emulate the production Toolkit functions, (2) to provide a development environment that emulates the production environment to support development and test, (3) make both functions and environment easy to use, and (4) most importantly, allow for a smooth transition of science software from the SCF to the production environment, during the integration and test phase.

The ECS science software developer will use the Toolkit to access the production environment and services, or their emulation. The Toolkit routines are divided into two classes:

a. Mandatory:

In order to access production services such as scheduling and messaging services in a consistent way, to avoid duplication of science software development effort, and to assure portability across computing platforms, usage of a subset of the Toolkit functions is required. These include functions that deal with file I/O, error message transactions, process control, ancillary data access, spacecraft ephemeris and attitude, and time and date transformations. The use of these tools will be enforced through automatic checks at integration time at the DAACs.

b. Optional:

Other useful functions required by developers, such as those involving celestial body positions, coordinate transformations, math libraries, physical constants, and graphics support, will be provided by the Toolkit. The use of these services is optional, but is encouraged. Science software developers who use alternative solutions will be required to deliver the source code (Portable Operating System Interface for Computer Environments (POSIX) compliant) for the replacement services as part of the algorithm delivery. Prohibited and allowed system calls are the subject of Appendix K on POSIX.

The Toolkit will serve to insulate science software from the Science Data Processing (SDP) software, and to provide a development environment that emulates critical SDP–functions. In most cases, a complete simulation of the DAAC SDP System will not be required. The Toolkit will help ensure code portability as the algorithm is ported from development hardware, through the DAAC system, and through potential hardware changes as the ECS matures. To do so effectively, the Toolkit will provide for limited access and control to system level resources, including processes, shared memory, and I/O capabilities. Where control of such resources is necessary (e.g., shared memory allocation), the Toolkit will provide a set of routines through which the application must obtain those services. This partitioning and layering of operating system services allows the Toolkit to work on behalf of the Data Processing subsystem in allocating, de–allocating, and making use of system–wide shared resources. The Toolkit will also

serve to minimize code development by providing common functionality required across the ECS community, such as geolocation.

It is essential to understand the concepts that distinguish the SCF development environment from the production environment. While the science software and interface to the SDP Toolkit are preserved in both environments, there are slightly different implementations and behavior in the Toolkit functions and peripheral components (e.g., shell level development external to the product generation executive (PGE) and testing tools). As far as the calling sequences themselves go, these differences are transparent to the science software developer, i.e., the calling sequences in the SCF and production environment versions are identical. Some setup of the underlying environment will be necessary at the SCF, as explained in Section 4.2 below. This setup should not affect the code itself.

4.2 SCF Development Environment

4.2.1 Introduction

This User's Guide describes tools that were designed to function in the production environment. For this reason, certain assumptions were made during their design process which will affect the operation of these tools in the SCF environment. It is the primary purpose of this section to identify those areas where extra measures will need to be taken, on the part of the SCF developers, to compensate for the differences in the two environments. To assist with this effort, utilities that are being developed to support ECS internal testing will be made available to SCF developers after they are developed and tested. These utilities are expected to prove useful to Product Generation Executive (PGE) script development as well as to the integration and testing processes. Aside from supplying the production environment emulation services necessary to fully utilize the SDP Toolkit, these utilities will also provide an integrated environment to facilitate the specification and execution of test scenarios. Production environment emulation utilities will evolve over time as the architecture and system design of the ECS progress.

It is also the intent of this section to impart to the SCF developers our view of how science software development should be undertaken at the SCFs where the Toolkit is concerned. The intent here is merely to present our views and not to impose guidelines on the actual development process. If a future implementation of ECS, for example, allows for standard product production at the SCFs, usage of the Tools and utilities presented in this document should not impede but aid algorithm development.

4.2.2 File Management

In the production environment, product files coming from the system archive are designated by Earth Science Data Type (ESDT). In order for science software to access staged files, a scheme for translating internal software identifiers into actual physical identifiers has been established (Section 6.2.3). The same holds true for the SCF environment since the same I/O tools will be used to access these files from within the science software. The main difference being that in the production environment, these filename references are resolved when a PGE is queued for execution. Since the production environment will not be part of the SCF environment, a

mechanism was devised to substitute for this functionality. This mechanism, known as a process control file (PCF), involves the creation of an external mapping of logical identifiers to physical file names according to the specifications for such a mapping. In this fashion, the software interface is consistent in both the SCF and DAAC run environments.

Some other notes regarding files concern the support for a one-to-many, logical-to-physical relationship among Product Input files. While this functionality is supported by the Toolkit, there are several guidelines that must be observed when defining these associations through the PCF mechanism. The first of these requires that files can only have more than one instance if they are entered into the section of the PCF labeled PRODUCT INPUT FILES. Since the logical identifier is static for files of this type, an instance number is required by the Toolkit, when references are made, to distinguish amongst several files in the group. In order to ascertain the number of instances associated with a logical identifier, you must invoke the Toolkit function that provides this information (Section 6.2.3.2). Second, the order in which associated Product Input files is retrieved, using a sequentially increasing instance number, is the same order in which they are presented in the PCF, e.g., an instance number of 3 indicates the third associated Product Input file defined in the PCF. Third, associated Product Input Files are those which possess the same logical identifier and appear in succession in the PCF. Lastly, the instance number is NOT directly related to the sequence number that appears at the end of the Record Field in the PCF for each Product Input file (Appendix C) -- that sequence is the inverse of the actual presentation in the PCF, such that the last entry in an association has Record Field = 1, the second to last has Record Field = 2, while the first entry has its Record Field equal to the number of entries in the association.

Until more is known about the ability to request that Product Input files be staged (loaded to disk and updated in the PCF) in a specific order, we recommend that you NOT anticipate that any specific ordering will exist in the production environment. Rather, always examine the file attributes (metadata) to ascertain the specifics about the Product Input file before referencing it.

At the SCF, users must populate entries in the PCFs they intend to use during testing of PGEs. At the DAAC, the PCF used in production is populated by the production system at runtime, based on data dependencies and scheduling rules communicated to the DAAC Science Software Integration and Test (SSI&T) team.

4.2.3 Runtime Configuration

To support a wide range of testing scenarios, some runtime parameters may be required to modify the behavior of the PGE under certain conditions. The SDP Toolkit contains the routines necessary to access the values of these parameters during runtime, provided that an external mapping of logical identifiers to actual values has been performed according to the specifications for this type of mapping.

In the production environment, dynamic control of these parameters occurs through a client interface that constructs production requests; the parameter changes resulting from such activation of this mechanism override the default mappings maintained in the production environment. There are also certain such runtime parameters that are dynamically determined immediately prior to PCF creation within the production system.

4.2.4 PGE Script Development

PGE scripts build the logical framework around the executables that produce the science products. It is our view that these scripts should be created by SCF science software developers, perhaps with guidance from the DAAC. It is also our understanding that the same Product Generation Executive (PGE) script will be delivered to the DAAC SSI&T team with little or no modifications required. In order to achieve this, the actual script should ideally be developed using a POSIX.2 conforming shell language. If at the time of development such a shell is not supported for all the approved platforms, development may proceed by using the standard Bourne shell (or other shell language approved by the ECS Project) on those platforms lacking a POSIX.2 implementation.

The actual PGE script as initiated in the production environment will not take arguments from the command line. Instead, script calls to command versions of some 'Process Control' tools (see Section 6.2.3) will provide for the retrieval of pertinent runtime information. Likewise, the routine versions of the same tools should be used to obtain runtime information from within the executables, rather than passing this information through the shell interface. This allows for easier configuration of executables within the PGE script should modifications be required at some point in the future. This scheme is possible since the executable interfaces, files and runtime parameters, are defined and maintained external to the PGE script in the production environment. It is to the SCF developer's benefit to adhere to this convention wherever possible, to ensure portability of software into the production environment.

To support the startup and housekeeping needs of the SDP Toolkit; a Toolkit shell command has been developed which performs the necessary initialization and termination procedures. This shell command accepts a PGE script as input, assuring that execution of the PGE occurs between the initialization and termination phases of the Toolkit. This shell command is similar to that which will be run in the production environment to guarantee the proper activation and deactivation of the Toolkit. It is recommended that SCF developers utilize this tool when conducting their testing.

When testing for the exit status of an executable within the PGE, only two values should exist : (0) for success and (1) for failure. This will require the executable developers to invoke the library exit call with the appropriate value as the final statement in their software. The same holds true for the exit status of the PGE with the exception that the shell command 'exit' is invoked instead.

The Toolkit will support the following script languages: Bourne shell, C shell, Korn shell, POSIX and the Perl language. However, certain system calls within these languages are prohibited (as are such calls from the PGE executables), most notably, any file system activity other that read/write.

4.2.5 Scheduling and Execution of PGEs

As was previously stated, scheduling or queuing of PGEs via the data production processing subsystem will not be part of the SCF. However, developers should be able to generate scenario scripts for different PGEs that will emulate the execution of PGEs within the DAAC

environment. With each scenario script tailored to execute a single PGE for specific set of conditions, a superscript that activates several scenario scripts could be used to perform the execution of multiple PGEs, further enhancing the emulation.

4.2.6 Error/Status Message Creation and Use

The 'smfcompile' utility provided in the SDP Toolkit (see Section 6.2.2) contains all the required functionality for defining and maintaining error and status codes, user messages and associated action messages. This tool, while only used in the SCF development environment, will fully support the suite of 'Error and Status Reporting' tools at both the SCF and the production environment.

Designed to support modular program development, this utility allows for separating the task of defining status codes and messages from the actual software development task. In fact, the process of defining these codes and messages may even be performed in the design stage, only later to be referenced during software development. This is an especially useful arrangement if action messages are to be incorporated into the status codes. This way, someone other than the programmer can decide the action that needs to be taken when a certain error, or status condition occurs.

While we do not endorse the creation of a separate Status Message File (SMF) for each routine/function, etc., we do recommend that SMF file creation follow the logical partitioning of software modules. So for a related set of routines, or even a small program, there might only be one SMF that defines the status codes and messages returned by those routines.

The format for defining a status code mnemonic is intentionally free–form to allow the developer to create and reference status codes that convey some meaning when writing and visually inspecting the code.

4.2.7 Error/Status Log Monitoring

In the production environment, an error/status log file will be opened just before execution of the PGE. This will be accomplished through an 'Initialization' command invoked by the production environment. This tool and its associated 'Termination' command, were delivered with the Toolkit 4 release of the SDP Toolkit. Developers can insert these tools at the beginning and end respectively of a superscript that encapsulates their PGE script. However, it would be preferable to use the Toolkit Shell command, since it already calls these commands and provides for the encapsulation of a PGE script. If the emulation utility is used, the scenario scripts that it generates will automatically incorporate these tools.

The actual log file name will most likely be influenced by system parameters in the production environment. The easiest mechanism for accomplishing this in the SCF environment will be through the assignment of some file name to the appropriate Record Field, in the process control file (PCF), for each status log. The emulation utility may allow for the log file to be defined through the file management services.

Through the emulation utility, the actual name of the log file could be conveyed to the user on the host platform's console, or through some other convenient means. The user will most likely initiate a scrolling output of the log file to a terminal window, to monitor the progress of PGE execution.

The 'Termination' command mentioned earlier will be responsible for closing the log file and dispatching it to some pre–defined location as specified by a Runtime Parameter in the PCF.

4.2.8 Parallel Processing Issues

While the majority of the software to be designed at the various SCF locations will be sequential in nature, due to its direct dependency on data, some portion of that software lends itself to being processed in a parallel fashion. This is especially true of those processes that share a common set of input data, but which have no interdependencies themselves.

Unfortunately, the system architecture that would define the ability to execute portions of a PGE on non–host platforms (i.e., a massively parallel machine) in the production environment has not yet been determined. Until such an architecture is defined, if at all, developers will only be able to test concurrent execution of executables on a single host. If a requirement for this type of processing is derived, the Toolkit will be configured to work in that environment.

4.2.9 Configuration Management

The importance of having a robust configuration management (CM) tool for a project of this size cannot be overstated. From SDP Toolkit development to science software development and integration, the use of this tool will control the version of software to support the continuous development and execution of production software.

After careful evaluation of several CM products, the ClearCase tool from Atria Software was chosen to support the internal software development, during site CM and maintenance and operations (M&O) CM requirements analysis. It is recommended that compatibility and interoperability benefits be explored.

4.2.10 Distributed Computing Environment (DCE) Issues

With the advent of distributed computing, an ever-increasing amount of single process execution will be performed across multiple machines instead of the more typical scenario of many processes running on one machine. While this technology may someday help to improve the efficiency of your process, and at the same time take advantage of underutilized processors, the constraints that it places on the ECS system architecture preclude the use of certain Distributed Computing Environment (DCE) features. Among them is the use of remote procedure calls (RPC's). The creation of RPC's to perform some segment of processing for a science algorithm is such a customized task that its interface cannot be generalized into some extended SDP Toolkit functionality. Since it is the Toolkit's charter to isolate the science software from the system architecture, the SDP Toolkit's inability to mask this feature prohibits its direct usage in the actual production software. For this reason, the direct use of RPC’s will not be allowed in the algorithm software developed by the instrument teams.

We note that an interface that makes RPC's indirectly available to science users through a client interface is being considered in revisions of the ECS architecture. This interface may become an extension of the Toolkit and will allow the algorithmic access of data through parameter–based searches. The details and limitations of this interface are not available at the time of this document.

4.3 Test and Simulation Data Access

The Toolkit provides tools to access all external data files required for science processing development and execution. There are tools to provide the read functions to all data types: L0, ancillary data, calibration coefficient files, standard products, etc.

Clearly test data must be accessed through the tool with the same Toolkit interface as in the production environment. In general, the Toolkit will aim to provide low level "write" functions to match the "read" functions so that the users may develop their own test data sets to the format required. Although there are currently no requirements that the Toolkit supply these new "write" tools, it is expected that they will be required for adequate testing within the production environment. In certain cases, such as platform orbit and attitude simulation, the Toolkit may provide specially prepared test data sets.

For example, the L0 data write tool will provide a function to write data into a file formatted as the packet based structure expected from EOSDIS Data and Operations System (EDOS). In this example, the "write" routine would require that the science data is provided by the user so that the test data set may be tailored to the user needs.

For the case of dynamic external auxiliary data (e.g., Special Sensor for Microwave Imaging (SSM/I) water vapor data) software may be provided to preprocess external data into any internal format used in the production environment, so that consistent data sets for testing may be developed by the user as required.

In the current implementation, EOS AM, EOS PM, EOS AURA and Tropical Rainfall Measuring Mission (TRMM) orbit and attitude simulation are supplied with the Toolkit. A packetizing tool Level 0 instrument science data simulation (which can be used in conjunction with the orbit simulator) is under development. A digital chart of the world (DCW) data base and a celestial body ephemerides are also provided with the current delivery.

4.4 Language Bindings and Advanced FORTRAN Considerations

The calling sequences in this document are provided in the C language with FORTRAN calling sequences provided in addition for most tools. The toolkit may now be built with the C++ compiler. The C++ library contains FORTRAN bindings (this means that the C++ Toolkit libraries can be called from FORTRAN).

a. The SDP Toolkit is designed in C, with most of the FORTRAN interface provided via inter–language bindings. In cases where there is no obvious relationship between FORTRAN and C calls, i.e., C pointers and structures, bindings will have to be done

carefully so as not to cause processing impairment. Note that there are no such tools in the current implementation.

b. The question of computing speed has a strong effect on the design of FORTRAN tools. Some tools, such as Level 0 I/O tools, need to be as fast as possible—the extra layer of binding from C to FORTRAN may slow the processing to the point that the tool is unusable. Therefore, a subset of the SDP Toolkit is designed as FORTRAN—only; i.e., not bound to C, for this reason. The user interface will not change, however.

c. FORTRAN77 is currently the highest level of FORTRAN that has a POSIX standard. However, many features of FORTRAN90 that are not present in FORTRAN77 are desired for inclusion in the Toolkit. These FORTRAN90 features include pointers and structures. This may mean that there will be two sets of FORTRAN calling sequences, one for 77 and one for 90. There are no FORTRAN90 only constructs in the current implementation.
   The tools compile in both FORTRAN77 and FORTRAN90

e. The only Ada support offered by the Toolkit is in the generation of Status Message Files by the 'smfcompile' utility.

4.5 Thread-Safe Issues

The PGS Toolkit may now be built in either Threadsafe or non-Threadsafe mode. The user may NOT use the Threadsafe library (libPGSTK_r.a) for a non-threaded PGE applications and likewise the user may NOT use the non-Threadsafe library (libPGSTK.a) for a threaded PGE application. Intermixing libraries and executables will cause undefined results.

The user API remains the same for both the Threadsafe and non-Threadsafe Toolkit. All Toolkit Threadsafe code is internal and hidden from the user. The Toolkit adheres to POSIX.1c compliancy. Therefore, the pthread library (libpthread.a) is used. Using another thread library while using the Threadsafe version of the Toolkit is strongly discouraged as undefined and untested results may occur.

The COTS packages that Toolkit uses (ODL, etc.) are not Threadsafe. Therefore, it is highly recommended that functions in Toolkit groups that call a COTS package should be called from the same thread. The groups that would not be considered Threadsafe are CBP, CSC, CUC, AA, GCT, DEM, and MET, and HDF (HDF-EOS). Calling any of these groups from multiple threads will lead to undefined results (i.e. core dumps).

It was also discovered during testing that great care must be taken while writing multi-threaded programs. Since more system resources are taken when using multiple threads, hidden coding oversights can become serious errors. For example, failing to close a file: in a multi-threaded program, a file may be opened many different places, and high numbers of open files could will eventually lead to the maximum number of files being opened; and an error will result.

Great care must also be taken to ensure that all data variables are local. For example, global variables can be used and modified by any active thread. Since each thread has a distinct and different purpose the globals, will be set to the necessary value for that specific thread. The next

thread accessing a global will probably error out due to erroneous data values. This is the exact problem with the COTS packages.

Limiting the number of threads that make Toolkit calls will aid in receiving the expected results. Running any threads, in general, can be a resource drain on a computer; and running a Threadsafe Toolkit can multiply the resource drain on a machine. Testing for the Threadsafe Toolkit, which had multiple threads only calling Toolkit functions, revealed that performance was better with a limited number of threads.

Below is an example of the use of Toolkit functions in a multi-thread program.

```
/*
 * thread.c
 *
 * Demonstrate how only one thread is allowed to call
* ALL functions in the Toolkit and the remaining threads
 * are restricted as to which groups they can call.
 */
#include <pthread.h>
#include <stdio.h>
void *ThreadA (void *arg)
{
/**
Thread A calls any and all functions in the Toolkit
**/
   return NULL;
}
void *ThreadB (void *arg)
{
/**
Thread B makes Toolkit calls but does NOT call
CBP, CSC, CUC, AA, GCT, DEM, or MET.
**/
```

```
   return NULL;
}
void *ThreadC (void *arg)
{
/**
Thread C makes Toolkit calls but does NOT call
CBP, CSC, CUC, AA, GCT, DEM, or MET.
**/
   return NULL;
}
void *ThreadD (void *arg)
{
/**
Thread D makes Toolkit calls but does NOT call
CBP, CSC, CUC, AA, GCT, DEM, or MET.
**/
   return NULL;
}

int main (int argc, char *argv[])
{
   pthread_t   threadA;
   pthread_t   threadB;
   pthread_t   threadC;
   pthread_t   threadD;

   pthread_create (&threadA, NULL, ThreadA, NULL);

   pthread_create (&threadB, NULL, ThreadB, NULL);
```

```
    pthread_create (&threadC, NULL, ThreadC, NULL);


    pthread_create (&threadD, NULL, ThreadD, NULL);


    pthread_exit (NULL);
    return 0;
}
```

Again, any calls of SDP Toolkit groups that call COTS packages should be called in the same thread.

Although all COTS libraries that are called from the Toolkit are assumed to be non-Threadsafe and will be locked with a mutual exclusion (mutex) lock this does not make the packages themselves Threadsafe.

The Threadsafe PGS Toolkit library may be called from any thread of a multi-threaded application, but it does not manage scheduling of threads by a calling program, nor does it do anything to insure thread safety in routines that it calls. These programs and libraries must themselves assure the correctness of the sharing between threads.

An application program is responsible for managing its own shared memory buffers. If multiple threads are writing and/or reading to and/or from shared areas of memory, the Threadsafe Toolkit library cannot guarantee that the results will be correct. For example, if an application program stores results from one Threadsafe PGS Toolkit call in shared memory in one thread and another thread expects to read those results the Threadsafe Toolkit can not manage this type of synchronization. It is the responsibility of the application program to manage shared memory/file access.

The Threadsafe PGS Toolkit library accesses disk storage through the operating system, so for multi-threaded programs the Threadsafe Toolkit library provides whatever semantics the operation system provides. Hence, when multiple threads read and write to the same area on disk the Threadsafe Toolkit does not guarantee consistent results beyond that provided by the operating system. The Threadsafe Toolkit can guarantee that each read and write will be completed correctly, but the order of completion is unspecified, and might vary from run to run or from platform to platform.

C library functions that are called by the Toolkit that are not Threadsafe will be replaced with the _r counterpart. It is an application's responsibility to make sure that other libraries are called in an appropriate manner. For instance, if the MPIO and/or MPI libraries are not MT-Safe then the application should not use the MPIO file access driver. It is beyond the scope of the Threadsafe Toolkit configuration to determine when supporting libraries are Threadsafe.

The Threadsafe PGS Toolkit is not interprocess-safe. Two processes cannot simultaneously access a PGS Toolkit file, so no attempt is made to prevent deadlocks in the Threadsafe Toolkit by resetting state information with pthread_atfork(). Do not call Threadsafe Toolkit functions from a child process.

The Threadsafe PGS Toolkit serializes accesses to the library, each API call is atomic. If an application needs a sequence of operations to be atomic (e.g. Read, Modify, Write), the application code must provide the appropriate concurrency protocols.

The Threadsafe PGS Toolkit uses the same PCF for all threads in the PGE. All current rules for the PCF apply.

The Threadsafe PGS Toolkit will produce one set of SMF Error/Status files for the threads in the PGE. Each entry in the LogStatus file will be followed by a Thread ID (TID) number which will allow the user to trace a threads progress.

There is only one difference in return values in the Threadsafe API and the non-Threadsafe API. The Threadsafe Toolkit API may return PGSTSF_E_GENERAL_FAILURE. This states that there was a severe problem initializing, locking, or accessing keys. It is recommended that the application program exits upon receiving this return value.

5. Toolkit Installation and Maintenance

5.1 Installation Procedures

5.1.1 Release 9 SDP Toolkit Release Notes

5.1.1.1 Multiple Architecture Support

The Toolkit has the option of being installed with simultaneous support for multiple architectures. This means that it is no longer necessary to install a separate copy of the Toolkit for each host architecture to be supported. Instead, a single copy of the Toolkit, installed on a file server in a networked environment, may serve multiple hosts of different architecture types.

Running concurrent tasks on the Toolkit is possible, but it requires that each process be configured so that all output files, including Toolkit log files, are written to a separate area to avoid collisions. This is done by using a private customized Process Control File (PCF) for each concurrent task. Please refer to Appendix C for more information. Note that any such PCF MUST contain all of the entries in the master template PCF for proper Toolkit functioning.

The directory structure of the Toolkit was revised to allow multiple architecture support. Subdirectories of the Toolkit home directory are now as follows:

bin	binary and script executables	Note 1
database	data resource files used by the Toolkit	Note 1
doc	documentation	
include	header files	
lib	the Toolkit Library	Note 1
message	message files used by the error/status tools	
obj	object files used to build the Toolkit Library	Note 1
runtime	runtime files	Note 2
src	source code	
test	test area	

Note 1:

The directories bin, database, lib, obj and objcpp all contain architecture-specific files residing in subdirectories named for the architecture. One such subdirectory will be created for each run of the installation script on a given architecture. Toolkit environment variables are set by the environment scripts to automatically map to the appropriate directories.

The database directory additionally contains a subdirectory named common for data files shared by all architectures.

Note 2:

The directory runtime contains data files shared by all architectures. In addition, it contains one subdirectory for the each of the supported architectures. These subdirectories are for architecture-specific runtime files. Currently the only file distributed in these subdirectories the default Process Control File (PCF) PCF.relB0, which contains architecture-specific pathnames. Several files generated at runtime by a PGE (e.g. log files) are set by default (in the PCF) to be created in this directory as well.

5.1.1.2 DAAC Toolkit Support

The Toolkit supports DAAC as well as SCF sites. A single distribution file supports all sites. The type of Toolkit built is determined by command line options to the installation script.

5.1.1.3 Support for the IRIX 6.5 Operating System

The Toolkit now fully supports the SGI IRIX64 Operating System. Under IRIX64 there are three Application Binary Interfaces (ABI). The Toolkit treats each of these ABIs as a separate architecture. The table below gives the formats:

ABI	compiler flag	Toolkit name
old-style 32 bit	-32	sgi
new-style 32 bit	-n32	sgi32
64 bit	-64	sgi64

The old-style 32-bit format is backwards-compatible with 32-bit SGI platforms. The other formats run only under IRIX 6.x.. Please note that SGI plans to drop support for old-style 32-bit format, it is therefore strongly recommended that all users migrate to new-style 32 bit or 64-bit mode. Also, ECS DAAC facilities no longer support old 32 processing on the SGI.

5.1.1.4 HDF Integration

The Toolkit installation procedures include a section that covers the installation of The HDF Group's HDF file access packages, HDF4 and HDF5. HDF has been adopted as the standard data format for EOSDIS Core System product generation, archival, ingest, and distribution capabilities.

Currently, HDF4 is only needed in order to build and use the Digital Elevation Model (DEM), Metadata (MET), Ancillary Access (AA) tools, and EPH/ATT tools. In addition, MET tools require HDF5. If you do not plan to use these tools, the HDF4 and/or HDF5 installation section may be skipped. In future releases, we expect greater integration of the Toolkit with HDF.

An installation script for HDF4 and HDF5 is included as part of the main SDP Toolkit distribution. It is provided to simplify the installation of HDF as much as possible, greatly reducing the number of steps in The HDF Group's own installation procedure. As of Release 9, the toolkit uses HDF-4.2.13 and hdf5-1.8.19. The HDF distributions themselves are located in compressed tar files, called HDF-4.2.13.tar.gz and hdf5-1.8.19.tar.gz which must be downloaded

separately along with the ZLIB tar file zlib-1.2.11.tar.gz, JPEG tar file jpegsrc.v9b.tar.Z, and SZIP tar file szip2.1.1.tar.gz.

With a full installation, HDF requires approximately 60 Mb of disk space. After the installation files are cleaned up. They may be installed in any location; i.e., they do not have to be stored under the SDP Toolkit home directory. The disk partition where HDF4 and HDF5 are installed should have about 120 Mb of free space.

5.1.1.5 HDF-EOS Integration

The toolkit installation procedures now include a section which covers the installation of HDF-EOS and HDF-EOS5, standalone packages that may be used in conjunction with the toolkit. They implement the EOS standard methods for accessing HDF format files. Three interfaces are provided: Point, Swath and Grid. Please refer to the HDF-EOS and HDF-EOS5 User's Guide for more information. The distribution file for HDF-EOS and HDF-EOS5 are available from the same ftp server where the toolkit distribution files are located.

The toolkit HDF-EOS and HDF-EOS5 installations are only available if the toolkit is built with HDF support. It handles the details of unpacking the distribution file, setting HDF dependencies, and running the HDF-EOS installation script.

Currently, HDF-EOS (HDF4 based) is only needed in order to build and use the Digital Elevation Model (DEM) tools. If you do not plan to use these tools, the HDF-EOS installation section may be skipped.

HDF-EOS (or HDF-EOS5) may also be installed manually, either before or after the toolkit is installed. HDF4 (or HDF5) must be installed before installing HDF-EOS (or HDF-EOS5).

5.1.2 To Install the SDP Toolkit from a Disk–Based Tar File

5.1.2.1 Preliminary

If HDF4 and HDF5 are to be installed at this time (recommended), you must first download the HDF4 distribution file HDF-4.2.13.tar.gz, zlib-1.2.11.tar.gz, jpegsrc.v9b.tar.Z, szip2.1.1.tar.gz, and hdf5-1.8.19.tar.gz before proceeding. They may be loaded into any directory on your system, i.e. they need not reside in the SDP Toolkit home directory. The same applies to the HDF-EOS and HDF-EOS5 distribution files HDF-EOS2.20.v1.00.tar.Z and HDF-EOS5.1.16.tar.gz, if you plan to install HDF-EOS (recommended) while installing the toolkit.

Important HDF Note:

The toolkit-supplied HDF installation scripts contain various platform-specific patches and bug fixes that allow HDF4 and HDF5 to be successfully installed on all platforms supported by the toolkit. In most cases, both the libraries and utilities are built. Also, the script automatically sets up the installed HDF directories so that the Toolkit can find them.

Because of these factors, we strongly recommend that even if you already have HDF-4.2.13, zlib-1.2.11, jpegsrc.v9b , szip2.1.1, and hdf5-1.8.19 installed, you RE-INSTALL HDF AT THIS TIME, using the toolkit-supplied HDF installation scripts.

Historical Note:

Please note the acronym PGS (Product Generation System) is used throughout the toolkit software in place of SDP. This is for historical reasons: the name was changed as of Release 3 of the toolkit. We regret any confusion this may cause.

5.1.2.2 Unpacking the Distribution File

1. Select a location for the SDP Toolkit directory tree. It should be on a disk partition with at least 80 Mb of free space. If you plan to install HDF in the same partition, you will need at least 140 Mb of free space. If you plan to install support for multiple architectures, you will need about 20 Mb Toolkit space + 60 Mb HDF space for each additional architecture supported.

   Multiple Architecture Support Note

   As previously mentioned, it is now possible to build the toolkit with support for multiple architectures. The distribution file need only be unpacked once, to support all architectures. If the toolkit is to be built with multiple architecture support, the area chosen to unpack the distribution should be on a network file system accessible from all hosts to be supported. (Please note that the SGI supports three different architectures. So, if building a multiple architecture installation to support the SGI only, the file system need not be accessible across the network.)

2. Copy the file SDPTK5.2.20v1.00.tar.Z to the target directory by typing the command:

   cp SDPTK5.2.20v1.00.tar.Z <target-dir>

   where <target-dir> is the full pathname of your target directory.

3. Set your default directory to the target directory by typing the command:

   cd <target-dir>

4. Uncompress this file and extract the contents by typing the command:

   zcat SDPTK5.2.20v1.00.tar.Z | tar xvf -

   This will create a subdirectory of the current directory called TOOLKIT. This is the top-level toolkit directory, which contains the full toolkit directory structure.

5.1.2.3 Starting the Installation Procedure

1. Set your default directory to the top-level toolkit directory by typing the command:

   cd TOOLKIT

   Starting with 5.2.20 version TOOLKIT can be auto configured and installed like HDF4,

HDF5, HDF-EOS2, and HDF-EOS5. If you prefer to install TOOLKIT and related software using auto configure features please see README-AUTOCONF file in the doc directory. The direction for autoconf installation of HDF-EOS2 and HDF-EOS5 are provided in the file AUROCONF_INSTALL in the doc directory of their source code distributions.

Multiple Architecture Support Note:

The toolkit installation script must be run once for each of the architectures to be supported. To do this, simply login to the desired host and set your directory to the top-level toolkit directory: <target-dir>/TOOLKIT. Then, proceed to run the installation script, starting at Step 2, below. The installation runs MUST be done ONE AT A TIME. Attempting to run concurrent installation procedures may cause errors.

2. Determine options for the toolkit installation script.

   Before running the toolkit installation script, you must determine the command line options appropriate for your site. These options are referred to in this section as <install-options>.

   These options tell the installation script such things as whether to build for SCF or DAAC, and whether to build for FORTRAN-90 compatibility, (FORTRAN-77 is the default). The table below gives the basic site options. Other options follow.

Site	FORTRAN	<install-options>
SCF	FORTRAN-77	(none)
SCF	FORTRAN-90	-f90
DAAC	FORTRAN-77	-daac
DAAC	FORTRAN-90	-daac -f90

   Please refer to part 1 of the Notes section, below, for information about platforms that currently support FORTRAN-90. When doing a FORTRAN-90 installation, the use of -fc_path option, (see below), is highly recommended.

   It is RECOMMENDED that you specify the name of the installation directory. When installing the Toolkit in a directory which is being auto-mounted or which is a link, the Toolkit may not be able to correctly determine the name of the directory where you are installing it. You can specify the name of the installation explicitly by adding the following to <install-options>:

   -pgshome <installation directory>

   where <installation directory> is the top-level Toolkit directory name (e.g.: /usr/local/TOOLKIT). Note that this option can NOT be used to specify an installation directory other than where the Toolkit has already been created in the steps prior to running the INSTALL script.

   If you wish to save the output of the installation run in a log file (RECOMMENDED), add the following to <install-options>:

-log <log-file>

Where <log-file> is the name of the log file.

If you wish to compile the Toolkit in debug mode add the following to <install-options>:

-dbug

This will replace the optimization flag "-O" with "-g" for all files compiled into the Toolkit library. This allows Toolkit routines to be viewed from within a source code debugger.

To install the C++ version of the library, libPGSTKcpp.a, you may use the -cpp option to specify that you want the C++ version. To do this, add the following to <install-options>:

-cpp

To ensure that the proper C++ compiler is found by the script, you may use the -cpp_path option to specify its location. To do this, add the following to <install-options>:

-cpp_path <C++-compiler-path>

Where <C++-compiler-path> is the full C++ compiler path for the desired C++ compiler (e.g. /user/loca/CC). This option should not be needed at most sites.

To ensure that the proper C compiler is found by the script, you may use the -cc_path option to specify its location. To do this, add the following to <install-options>:

-cc_path <C-compiler-path>

Where <C-compiler-path> is the full C compiler path for the desired C compiler (e.g. /user/loca/cc). This option should not be needed at most sites.

To ensure that the proper FORTRAN compiler is found by the script, you may use the -fc_path option to specify its location. To do this, add the following to <install-options>:

-fc_path <FORTRAN-compiler-path>

Where <FORTRAN-compiler-path> is the full FORTRAN compiler path for the desired FORTRAN compiler (e.g. /usr/local/pgf77). This is particularly advisable when using FORTRAN-90 (e.g. for f90 installation in a linux platform using Portland pgf compiler: –f90 –fc_path /usr/local/pgf90).

NAG FORTRAN-90 Note:

If using a NAG FORTRAN-90 compiler to build the toolkit library, add the -nag option to <install-options>, after the -f90 and-fc_path options. This will allow the toolkit to generate the proper C to FORTRAN bindings. This option should not be used when building the toolkit on an SGI. See the note, below.

SGI Multiple Architectures Note:

On the SGI (as of IRIX64 6.5), the default is to build the toolkit in 64-bit mode. The following table gives the option to specify the appropriate architecture to be built:

binary format	architecture	<install-options>
old-style 32 bit	sgi	(none)**
new-style 32 bit	sgi32	-sgi32
64 bit	sgi64	-sgi64

(**) The Toolkit may be installed in old-style 32-bit mode, but this is no longer the default and may not be supported in future releases as SGI will be dropping support for this format. To install the Toolkit in this mode, run the Toolkit without any special sgi flags and then when prompted for the sgi mode enter "sgi" (without the quotes) at the appropriate prompt.

SGI FORTRAN-90 Note:

On SGI and SGI Challenge platforms running IRIX 6.5 and earlier, the type of FORTRAN-90 compiler is automatically determined by the script. On the old style 32-bit SGI platform, the NAG compiler is used. On the 64-bit SGI Challenge platform, the compiler chosen depends on the binary architecture type selected.

The script will override the setting of the -NAG flag, if specified, because only the combination listed below will build properly. The following table shows which compiler is used for each architecture:

binary format	architecture	f90
old-style 32 bit	sgi	NAG
new-style 32 bit	sgi32	SGI
64 bit	sgi64	SGI

When the -NAG option is specified, it is a good idea to specify the f90 compiler location via the -fc_path option, ("Setting the FORTRAN compiler path", above), to ensure that the script uses the right compiler.

By default the Toolkit supports the C language and one of FORTRAN77 or FORTRAN90. The installation procedure, therefore, normally requires a FORTRAN compiler. If no FORTRAN compiler available the Toolkit may be installed without a FORTRAN compiler by specifying -no_ftn on the command line of the bin/INSTALL script.

Note that HDF still requires a FORTRAN compiler. In order the Toolkit to successfully install without a FORTRAN HDF must be installed independently (i.e. NOT from the Toolkit INSTALL script) (see HDF Installation Section, below).

If you have already installed The HDF Group's HDF4 package, you can specify the installation location explicitly. If you do so, the Toolkit installation procedure will not attempt to install HDF4, using the installation you have specified instead. To do this, add the following to <install-options>:

-hdfhome <HDF4 installation directory>

where <HDF4 installation directory> is the HDF4 directory which contains the bin/ lib/ and include/ sub-directories of the installed HDF4 package.

If you have already installed The HDF Group's HDF5 package, you can specify the installation location explicitly. If you do so, the Toolkit installation procedure will not attempt to install HDF5, using the installation you have specified instead. To do this, add the following to <install-options>:

-hdf5home <HDF5 installation directory>

where <HDF5 installation directory> is the HDF5 directory which contains the bin/ lib/ and include/ sub-directories of the installed HDF5 package.

If you have already installed ECS's HDF-EOS (HDF4 based) package, you can specify the installation location explicitly. If you do so the Toolkit installation procedure will not attempt to install HDF-EOS, using the installation you have specified instead. To do this, add the following to <install options>:

-hdfeos_home <HDF-EOS installation directory>

where <HDF-EOS installation directory> is the HDF-EOS (HDF4 based) directory which contains the bin/ lib/ and include/ sub-directories of the installed HDF-EOS package.

If you have already installed ECS's HDF-EOS5 (HDF5 based) package, you can specify the installation location explicitly. If you do so the Toolkit installation procedure will not attempt to install HDF-EOS5, using the installation you have specified instead. To do this, add the following to <install options>:

-hdfeos5_home <HDF-EOS5 installation directory>

where <HDF-EOS5 installation directory> is the HDF-EOS5 (HDF5 based) directory which contains the bin/ lib/ and include/ sub-directories of the installed HDF-EOS5 package.

WARNING: the installation procedure will not make any checks of the versions of any pre-installed packages you specify in this way. It is your responsibility to ensure that any such packages you specify in this manner are at the approriate version level for the version of the Toolkit being installed.

By default, the Toolkit installation is an interactive procedure. If you would like to run the installation in "batch" mode add the following to <install-options>:

-batch

Note that the installation procedure is not as flexible when run in this mode. Namely, when using the script to install HDF4, HDF5, HDF-EOS and/or HDF-EOS5, these packages will be installed under the TOOLKIT directory (i.e. the default locations for these packages). This behavior cannot be changed, although you MAY still specify the locations of pre-installed versions of these packages using the appropriate <install-options> (see above). Also, if you specify the -dbug switch the Toolkit, HDF and HDF-EOS will all be installed in debug mode. Finally if you attempt to install HDF (HDF4 or HDF5) and an installed HDF is found in the default location it will be deleted and the whole HDF (HDF4 or HDF5) package will be reinstalled. If you attempt to install HDF-

EOS (or HDF-EOS5) and an hdfeos (or hdfeos5) directory is found to exist in the default location it will be "re-used".

5.1.2.4 Run the Toolkit Installation Script

Please note that the installation script for this release of the toolkit requires user interaction. Because of this, it should NOT be run as a background task. The new installation script, bin/INSTALL, is actually a front end for eight other scripts: bin/INSTALL-HDF4, bin/INSTALL-HDF5, bin/INSTALL-HDFEOS-Wrap, bin/INSTALL-HDFEOS5-Wrap, bin/INSTALL-JPEG, bin/INSTALL-ZLIB, bin/INSTALL-SZIP and bin/INSTALL-Toolkit. Each of these scripts may be run with the -h option to display a usage message. In most cases, it will not be necessary to run any of these scripts directly from the command line.

To run the script, type the command:

bin/INSTALL <install-options>
where <install-options> is the list of options determined in the previous step.

The installation script will then run. It will output various startup messages, beginning with:

Toolkit Installation starting at <date/time>

If the platform is a 64-bit linux (or MacIntel) platform you will be asked to enter “lnx64 (or mac64)” or “lnx32 (or mac32)” for 64-bit or 32-bit installation respectively. Press return for default installation or enter lnx32 or lnx64 for Linux (mac32 or mac64 for MacIntel platforms), then press return installation.

The script will then output a message discussing the HDF requirement, after which it issues a prompt which gives you an opportunity to quit.

Continue installation [yes] ?

To continue the installation, press return.

ZLIB Installation Section

1. The script prompts with:

   Is zlib-1.2.11 installed at your site [no] ?

   If ZLIB is not installed, hit return and proceed to step 3, below.

2. If you already have the correct version of ZLIB installed, you may type 'y' and hit return. In this case, the script will ask where ZLIB is installed:

   Pathname where directory zlib-1.2.11 is located [<default>] ?

   Type in the full pathname and hit return. The script will check to make sure that ZLIB is really installed there. Please proceed to the toolkit Installation Section, below.

3. The script prompts with:

   Do you wish to install zlib-1.2.11 now [yes] ?

Hit return to continue.

4. The script responds with:

   Running the ZLIB Installation Script ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where zlib-1.2.11.tar.gz is located ?

   Please enter the correct location and hit return.

6. The script then asks where the ZLIB directory will be created. The default is <toolkit-home-directory>/zlib/$BRAND, where $BRAND is the toolkit architecture being built, given by the table in Note 2 of the NOTES section, below.

   Pathname where directory `zlib-1.2.11' will be created [<default>] ?

   If you want ZLIB installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue.

Multiple Architecture Support Note:

A copy of the ZLIB installation must be built for each of the architectures to be supported by this toolkit installation. We therefore recommend using the default ZLIB directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build ZLIB.

7. The script asks you to verify the information entered, prompting with:

   Continue [yes] ?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

8. This completes the interactive portion of the ZLIB installation. When the ZLIB section is complete, it outputs the message:

   ZLIB installation ending at: <date/time>

JPEG Installation Section

1. The script prompts with:

   Is jpeg-9b installed at your site [no] ?

   If JPEG is not installed, hit return and proceed to step 3, below.

2. If you already have the correct version of JPEG installed, you may type 'y' and hit return. In this case, the script will ask where JPEG is installed:

Pathname where directory jpeg-9b is located [<default>] ?

Type in the full pathname and hit return. The script will check to make sure that JPEG is really installed there. Please proceed to the toolkit Installation Section, below.

3. The script prompts with:

   Do you wish to install jpeg-9b now [yes] ?

   Hit return to continue.

4. The script responds with:

   Running the JPEG Installation Script ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where jpegsrc.v9b.tar.Z is located ?

   Please enter the correct location and hit return.

6. The script then asks where the JPEG directory will be created. The default is <toolkit-home-directory>/jpeg/$BRAND, where $BRAND is the toolkit architecture being built, given by the table in Note 2 of the NOTES section, below.

   Pathname where directory 'jpeg-9b' will be created [<default>] ?

   If you want JPEG installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue.

Multiple Architecture Support Note:

A copy of the JPEG installation must be built for each of the architectures to be supported by this toolkit installation. We therefore recommend using the default JPEG directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build JPEG.

7. The script asks you to verify the information entered, prompting with:

   Continue [yes] ?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

8. This completes the interactive portion of the JPEG installation. When the JPEG section is complete, it outputs the message:

   JPEG installation ending at: <date/time>

SZIP Installation Section

1. The script prompts with:

   Is szip2.1.1 installed at your site [no] ?

   If SZIP is not installed, hit return and proceed to step 3, below.

2. If you already have the correct version of SZIP installed, you may type 'y' and hit return. In this case, the script will ask where SZIP is installed:

   Pathname where directory szip2.1.1 is located [<default>] ?

   Type in the full pathname and hit return. The script will check to make sure that SZIP is really installed there. Please proceed to the toolkit Installation Section, below.

3. The script outputs:

   ```
   WARNING: Commercial users should obtain szip license

   if they intend to distribute their products with szip

   encoder. The szip decoder does not require license.
   ```

   and then prompts with:

   Do you wish to install full szip2.1.1 (encoder + decoder) [yes] ?

   Hit return to continue, or type ‘n’ and hit return. If you enter ‘n’ by default only the szip decoder will be installed.

4. The script responds with:

   Running the SZIP `(with/without encoding)` Installation Script ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where szip2.1.1.tar.gz is located?

   Please enter the correct location and hit return.

6. The script then asks where the SZIP directory will be created. The default is <toolkit-home-directory>/szip/$BRAND, where $BRAND is the toolkit architecture being built, given by the table in Note 2 of the NOTES section, below.

   Pathname where directory 'szip2.1.1' will be created [<default>] ?

   If you want SZIP installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue.

Multiple Architecture Support Note:

A copy of the SZIP installation must be built for each of the architectures to be supported by this toolkit installation. We therefore recommend using the default SZIP directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build SZIP.

7. The script asks you to verify the information entered, prompting with:

   Continue [yes] ?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

8. This completes the interactive portion of the SZIP installation. When the SZIP section is complete, it outputs the message:

   SZIP installation ending at: <date/time>

HDF4 Installation Section

1. The script prompts with:

   Is HDF-4.2.13 installed at your site [no] ?

   If HDF4 is not installed, hit return and proceed to step 3, below.

2. If you already have the correct version of HDF4 installed, you may type 'y' and hit return. In this case, the script will ask where HDF4 is installed:

   Pathname where directory HDF-4.2.13 is located [<default>] ?

   Type in the full pathname and hit return. The script will check to make sure that HDF4 is really installed there. Please proceed to the toolkit Installation Section, below.

3. The script prompts with:

   Do you wish to install HDF-4.2.13 now [yes] ?

   Hit return to continue.

   Then the script prompts with:

   Are you going to use external netCDF with your HDF4 applications [no]?

   If you intend to use external netCDF library with your hdf4 then enter ‘y’ otherwise hit return. If you answer ‘y’ then HDF4 will be installed with --disable-netcdf so that netCDF function in HDF4 are renamed (with prefix sd_), avoiding clash between name symbols of the internal and external netCDF packages.

   Then the script prompts with:

   Do you wish to configure HDF4 with SZIP[y] ?

Hit return if you wish the installed HDF4 have szip decoding (and/or encoding) capability.

4. The script responds with:

   Running the HDF Installation Script ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where HDF-4.2.13.tar.gz is located?

   Please enter the correct location and hit return.

6. The script then asks where the HDF4 directory will be created. The default is <toolkit-home-directory>/hdf/$BRAND, where $BRAND is the toolkit architecture being built, given by the table in Note 2 of the NOTES section, below.

   Pathname where directory 'HDF-4.2.13' will be created [<default>]?

   If you want HDF4 installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue.

Multiple Architecture Support Note:

A copy of the HDF4 installation must be built for each of the architectures to be supported by this toolkit installation. We therefore recommend using the default HDF4 directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build HDF4.

7. The script asks you to verify the information entered, prompting with:

   Continue [yes]?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

8. This completes the interactive portion of the HDF4 installation. When the HDF4 section is complete, it outputs the message:

   HDF installation ending at: <date/time>

HDF5 Installation Section

1. The script prompts with:

   Is hdf5-1.8.19 installed at your site [no]?

   If HDF5 is not installed, hit return and proceed to step 3, below.

2. If you already have the correct version of HDF5 installed, you may type 'y' and hit return. In this case, the script will ask where HDF5 is installed:

Pathname where directory hdf5-1.8.19 is located [<default>]?

Type in the full pathname and hit return. The script will check to make sure that HDF5 is really installed there. Please proceed to the toolkit Installation Section, below.

3. The script prompts with:

   Do you wish to install hdf5-1.8.19 now [yes]?

   Hit return to continue.

4. The script responds with:

   Running the HDF5 Installation Script ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where hdf5-1.8.19.tar.gz is located?

   Please enter the correct location and hit return.

6. The script then asks where the HDF5 directory will be created. The default is <toolkit-home-directory>/hdf5/$BRAND, where $BRAND is the toolkit architecture being built, given by the table in Note 2 of the NOTES section, below.

   Pathname where directory 'hdf5-1.8.19' will be created [<default>]?

   If you want HDF5 installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue.

Multiple Architecture Support Note:

A copy of the HDF5 installation must be built for each of the architectures to be supported by this toolkit installation. We therefore recommend using the default HDF5 directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build HDF5.

7. The script asks you to verify the information entered, prompting with:

   Continue [yes]?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

8. This completes the interactive portion of the HDF5 installation. When the HDF5 section is complete, it outputs the message:

   HDF5 installation ending at: <date/time>

HDF-EOS Installation Section

1. The script prompts with:

   Is HDF-EOS2.20v1.00 installed at your site [no]? [yes]?

   If HDF-EOS is not installed, hit return and proceed to step 3, below

2. If you already have the correct version of HDF-EOS installed, you may type 'y' and hit return. In this case, the script will ask where HDF-EOS is installed

   Pathname where HDF-EOS2.20v1.00 is installed [<default-path>]

3. The script prompts with:

   Do you wish to install HDF-EOS2.20v1.00 now [yes]?

   Hit return to continue

4. The script responds with:

   Installing HDF-EOS ...

   It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where HDF-EOS2.20v1.00.tar.Z is located?

   Please enter the correct location and hit return.

6. The script then asks where the HDF-EOS directory will be created. The default is <toolkit-home-directory>.

   Pathname where directory 'hdfeos' will be created [<default>]?

   If you want HDF-EOS installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue. If installing for an additional architecture, (refer to the Multiple Architecture Support Note in Step 1 of "Starting the installation procedure"), use the same directory as for the first instance of HDF-EOS - a single copy will support multiple architectures.

7A. Single-Architecture Installation

   If this is a single-architecture installation, or the first platform of a multiple-architecture installation, do this step. Otherwise proceed to step 7B.

   The script asks you to verify the information entered, prompting with:

   Continue [yes]?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

Proceed to step 8

7B. Multiple-Architecture Installation

If this is an additional platform in a multiple-architecture installation, i.e. the INSTALL script is being run again to add support for an additional architecture, (refer to the Multiple Architecture Support Note in Step 1 of "Starting the installation procedure"), proceed as follows:

The script asks you to verify the information entered, prompting with:

Continue [yes] ?

Hit return to continue. The script should respond with;

The directory hdfeos already exists.

[O]verwrite, [R]e-use or [Q]uit (default) ?

Type 'R' and hit return. The script will build HDF-EOS for the new architecture using the existing copy of the directory structure. Libraries and executables will be added to the architecture-specific subdirectories of the HDF-EOS 'bin' and 'lib' directories, respectively. Do NOT use the Overwrite option - it will clobber the previous architecture-specific installation(s).

8. This completes the interactive portion of the HDF-EOS installation. When the HDF-EOS section is complete, it outputs the message:

   HDFEOS installation ending at: <date/time>

   For information about user setup, as well as instructions for compiling and linking with HDF-EOS, Refer to the file README in the HDF-EOS 'doc' directory.

HDF-EOS5 Installation Section

1. The script prompts with:

   Is HDF-EOS5.1.16 installed at your site [no]? [yes]?

   If HDF-EOS5 is not installed, hit return and proceed to step 3, below

2. If you already have the correct version of HDF-EOS5 installed, you may type ‘y’ and hit return. In this case, the script will ask where HDF-EOS5 is installed

   Pathname where HDF-EOS5.1.16 is installed [<default-path>]

3. The script prompts with:

   Do you wish to install HDF-EOS5.1.16 now [yes]?

   Hit return to continue

4. The script responds with:

   Installing HDF-EOS5 ...

It may also output a few informational messages, depending on the installation options selected.

5. By default, the script looks for the distribution file in your current and parent directories. If the file is found in either of these locations, the script will continue to the next step. Otherwise, it will prompt with:

   Pathname where HDF-EOS5.1.16.tar.gz is located?

   Please enter the correct location and hit return.

6. The script then asks where the HDF-EOS5 directory will be created. The default is <toolkit-home-directory>.

   Pathname where directory 'hdfeos5' will be created [<default>] ?

   If you want HDF-EOS5 installed elsewhere, please enter the pathname at the prompt. Otherwise, simply hit return to continue. If installing for an additional architecture, (refer to the Multiple Architecture Support Note in Step 1 of "Starting the installation procedure"), use the same directory as for the first instance of HDF-EOS5 - a single copy will support multiple architectures.

7A. Single-Architecture Installation

   If this is a single-architecture installation, or the first platform of a multiple-architecture installation, do this step. Otherwise proceed to step 7B.

   The script asks you to verify the information entered, prompting with:

   Continue [yes]?

   Hit return to continue. The contents of the distribution file are then extracted into the specified location, and the installation procedure is run.

   Proceed to step 8

7B. Multiple-Architecture Installation

   If this is an additional platform in a multiple-architecture installation, i.e. the INSTALL script is being run again to add support for an additional architecture, (refer to the Multiple Architecture Support Note in Step 1 of "Starting the installation procedure"), proceed as follows:

   The script asks you to verify the information entered, prompting with:

   Continue [yes]?

   Hit return to continue. The script should respond with;

   The directory hdfeos5 already exists.

   [O]verwrite, [R]e-use or [Q]uit (default)?

   Type 'R' and hit return. The script will build HDF-EOS5 for the new architecture using the existing copy of the directory structure. Libraries and executables will be

added to the architecture-specific subdirectories of the HDF-EOS5 'bin' and 'lib' directories, respectively. Do NOT use the Overwrite option - it will clobber the previous architecture-specific installation(s).

8. This completes the interactive portion of the HDF-EOS5 installation. When the HDF-EOS5 section is complete, it outputs the message:

   HDFEOS5 installation ending at: <date/time>

   For information about user setup, as well as instructions for compiling and linking with HDF-EOS, Refer to the file README in the HDF-EOS5 'doc' directory.

Toolkit Installation Section

1A. SCF Installation

If the SCF version of the toolkit is being built (the default), the script outputs the messages:

Running the Toolkit Installation Script ...

The script prompts with:

Do you wish to install AA Tool [No]?

If you want AA tool installed, response with ‘y’.

If you do not have the correct version of HDF

The script prompts with:

No HDF Support…

If you need install AA Tool, please install HDF package…

SDP Toolkit installation cancelled….

If you have the correct version of HDF

The scrip responds with:

Running the Toolkit Installation Script with AA Tool

If you do not want AA Tool installed, hit return

The script responds with:

Running the Toolkit Installation Script without AA Tool

Toolkit installation script: INSTALL-Toolkit

Starting at: <date/time>

The SCF version of the toolkit library libPGSTK.a will be built

1B. DAAC Installation

If the DAAC version of the toolkit is being built (-daac option), the script outputs the messages:

Running the Toolkit Installation Script ...

Toolkit installation script: INSTALL-Toolkit

Starting at: <date/time>

The DAAC version of the toolkit library libPGSTK.a will be built.

1C. C++ Installation

If the C++ version of the Toolkit is being built (-cpp option), the script outputs the messages

The C++ version of the toolkit library libPGSTKcpp.a will be built

If the C++ install was successful, you should see the following messages:

INSTALL-Toolkit completed successfully at <date/time>

SDP Toolkit installation completed at <date/time>

NOTE: Currently the script is set up so that the C/FORTRAN version of the library will be built first with the C++ of the library libPGSTKcpp.a, afterwards.

2. The toolkit installation script outputs status messages as it goes, ending with:

INSTALL-Toolkit completed successfully at <date/time>

If an error occurred during the installation process, the last message will appear as:

INSTALL-Toolkit completed with errors at <date/time>

NOTE: If the installation was run with the -log option, the above messages will appear only in the log file, not on the screen.

3. Wait for completion messages. If no errors were encountered during either HDF or toolkit installation, the final script message is:

SDP Toolkit installation completed at <date/time>

Otherwise messages of the following form will appear:

INSTALL: Error: <error message>

SDP Toolkit installation canceled

4. Review the installation log.

Every attempt has been made to trap all possible installation errors and report them at the end of the installation process. Nonetheless, it is a good idea to review the installation log to verify that it completed without errors. If errors were noted, the log can help to identify precisely what went wrong. Please note that some warning

messages, (NOT fatal errors), may occur in the course of a normal successful installation run.

Note regarding the installation of AA tools:

Starting with SDPTK5.2.7 the user has opportunity not to install AA tools if they do not need them. The INSTALL script will prompt for User's response in installing (or ignoring) AA tools. The default is "N".

5.1.2.5 User Account Setup

Once the toolkit has been installed, the accounts of SDP toolkit users must be set up to define environment variables needed to compile and run code with the toolkit (see parts 2 and 3 of the Notes section 5.1.2.8, below). The type of setup depends on the user's login shell.

1A. C shell (csh) users:

Edit the SDP Toolkit user's .cshrc file to include ONLY ONE of the following two lines:

(EITHER:)

source <SDP-home-dir>/bin/$BRAND/pgs-env.csh

(OR:)

source <SDP-home-dir>/bin/$BRAND/pgs-dev-env.csh

where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script pgs-env.csh sets up all the variables discussed in part 3 of the Notes section, below, and it adds the toolkit bin directory to the user path.

The script pgs-dev-env.csh sets up all of the variables set by pgs-env.csh.cpp and adds the toolkit bin directory to the user path. In addition, it automatically sets up the compiler flag variables discussed in part 4 of the Notes section below, to work on any of the system environments listed in part 1 of the Notes section, below.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

C++ version of the scripts:

Edit the SDP Toolkit user's .cshrc file to include ONLY ONE of the following two lines:

(EITHER:)

source <SDP-home-dir>/bin/$BRAND/pgs-env.csh.cpp

(OR:)

source <SDP-home-dir>/bin/$BRAND/pgs-dev-env.csh.cpp

where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script pgs-env.csh.cpp sets up all the variables discussed in part 3 of the Notes section, below, and it adds the toolkit bin directory to the user path.

The script pgs-dev-env.csh.cpp sets up all of the variables set by pgs-env.csh.cpp and adds the toolkit bin directory to the user path. In addition, it automatically sets up the compiler flag variables discussed in part 4 of the Notes section below, to work on any of the system environments listed in part 1 of the Notes section, below.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

Note: setting of users PGS_PC_INFO_FILE shell environment variable:

The scripts pgs-env.csh and pgs-dev-env.csh will by default define the environment variable PGS_PC_INFO_FILE to have the value $PGSRUN/$BRAND/PCF.relB0 (see Note 3, below). Individual users should make local copies of this file and then set the environment variable PGS_PC_INFO_FILE to point to this local copy, which should be modified to suit the purposes of the user. This can be done by adding the following line to the users .cshrc file (e.g.):

setenv PGS_PC_INFO_FILE $HOME/PCF.relB0

This should be done in the .cshrc file AFTER the file pgs-env.csh (or pgs-dev-env.csh) has been used to establish the users Toolkit environment.

Note regarding path setup with pgs-dev-env.csh and pgs-dev-env.csh.cpp:

The scripts pgs-dev-env.csh and pgs-dev-env.csh.cpp also make available a variable called pgs_path. This can be added to the user's path to ensure that it accesses the directories necessary for the compilers and other utilities used to generate executable programs. It is not added to the user path by default, because in many cases it adds unnecessary complexity to the user path. To add pgs_path to the user path, modify the SDP Toolkit user's .cshrc file to include the following:

```
set my_path = ($path)                                    # save path
source <SDP-HOME-DIR>/bin/$BRAND/pgs-dev-env.csh         # PGS setup
set path = ($my_path $pgs_path)                          # add pgs_path
```

INSTEAD OF either of the two options listed at the beginning of this step. Note that it is the user's responsibility to set up his or her own path so that it doesn't duplicate the directories set up in pgs_path. Please also note that the pgs_path is added AFTER the user's path. This way, the user's directories will be searched first when running Unix commands.

1B. Korn shell (ksh) users:

Edit the SDP Toolkit user's .profile file to include ONLY ONE of the following two lines:

(EITHER:)

<SDP-home-dir>/bin/$BRAND/pgs-env.ksh

(OR:)

<SDP-home-dir>/bin/$BRAND/pgs-dev-env.ksh

where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script pgs-env.ksh sets up all the variables discussed in part 3 of the Notes section, below, and it adds the toolkit bin directory to the user path.

The script pgs-dev-env.ksh sets up all of the variables set by pgs-env.ksh and adds the toolkit bin directory to the user path. In addition, it automatically sets up the compiler flag variables discussed in part 4 of the Notes section below, to work on any of the system environments listed in part 1 of the Notes section, below.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

Note: setting of users PGS_PC_INFO_FILE shell environment variable:

The scripts pgs-env.ksh and pgs-dev-env.ksh will by default define the environment variable PGS_PC_INFO_FILE to have the value $PGSRUN/$BRAND/PCF.relB0 (see Note 3, below). Individual users should make local copies of this file and then set the environment variable PGS_PC_INFO_FILE to point to this local copy, which should me modified to suit the purposes of the user. This can be done by adding the following line to the users .profile file (e.g.):

```
set PGS_PC_INFO_FILE=$HOME/PCF.relB0
export PGS_PC_INFO_FILE
```

This should be done in the .profile file AFTER the file pgs-env.ksh (or pgs-dev-env.ksh) has been used to establish the users Toolkit environment.

Note regarding path setup with pgs-dev-env.ksh and pgs-dev-env.ksh.cpp:

The script pgs-dev-env.ksh.cpp and pgs-dev-env.ksh.cpp also make available a variable called pgs_path. This can be added to the user's path to ensure that it accesses the directories necessary for the compilers and other utilities used to generate executable programs. It is not added to the user path by default, because in many cases it adds unnecessary complexity to the user path. To add pgs_path to the user path, modify the SDP Toolkit user's .profile file to include the following:

```
my_path="$PATH"                                    # save path
```

<SDP-HOME-DIR>/bin/$BRAND/pgs-dev-env.ksh # PGS setup

PATH="$my_path:$pgs_path" ; export PATH # add pgs_path

INSTEAD OF either of the two options listed at the beginning of this step. Note that it is the user's responsibility to set up his or her own path so that it doesn't duplicate the directories set up in pgs_path. Please also note that the pgs_path is added AFTER the user's path. This way, the user's directories will be searched first when running Unix commands.

C++ version of the scripts:

Edit the SDP Toolkit user's .profile file to include ONLY ONE of the following two lines:

(EITHER:)

<SDP-home-dir>/bin/$BRAND/pgs-env.ksh.cpp

(OR:)

<SDP-home-dir>/bin/$BRAND/pgs-dev-env.ksh.cpp

where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The script pgs-env.ksh.cpp sets up all the variables discussed in part 3 of the Notes section, below, and it adds the toolkit bin directory to the user path.

The script pgs-dev-env.ksh.cpp sets up all of the variables set by pgs-env.ksh.cpp and adds the toolkit bin directory to the user path. In addition, it automatically sets up the compiler flag variables discussed in part 4 of the Notes section below, to work on any of the system environments listed in part 1 of the Notes section, below.

The environment variables will become available during all subsequent login sessions. To activate them for the current session, simply type one of the two lines listed above, at the Unix prompt.

1C. Bourne shell (sh) users:

Set up the required toolkit environment variables by appending the contents of the file

<SDP-home-dir>/bin/$BRAND/pgs-env.ksh

or the file

<SDP-home-dir>/bin/$BRAND/pgs-dev-env.ksh

to the end of the SDP Toolkit user's .profile, where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The environment variables will become available during all subsequent login sessions. To activate them, log out and then log back in.

Bourne shell (sh) users:

Set up the required toolkit environment variables by appending the contents of the file

<SDP-home-dir>/bin/$BRAND/pgs-env.ksh

or the file

<SDP-home-dir>/bin/$BRAND/pgs-dev-env.ksh

to the end of the SDP Toolkit user's .profile, where <SDP-home-dir> is the full path of the toolkit home directory, and $BRAND is an architecture-specific value for your host. Please refer to part 2 of the Notes section, below, to determine the correct value.

The environment variables will become available during all subsequent login sessions. To activate them, log out and then log back in.

Note: setting of users PGS_PC_INFO_FILE shell environment variable:

The scripts pgs-env.ksh and pgs-dev-env.ksh will by default define the environment variable PGS_PC_INFO_FILE to have the value $PGSRUN/$BRAND/PCF.relB0 (see Note 3, below). Individual users should make local copies of this file and then set the environment variable PGS_PC_INFO_FILE to point to this local copy, which should be modified to suit the purposes of the user. This can be done by adding the following line to the users .profile file (e.g.):

```
set PGS_PC_INFO_FILE=$HOME/PCF.relB0
export PGS_PC_INFO_FILE
```

This should be done in the .profile file AFTER the file pgs-env.csh (or pgs-dev-env.csh) has been included.

5.1.2.6 File Cleanup

Once the toolkit has been built and tested, you can delete certain temporary files and directories to save some disk space. Note that once these files have been removed, you will need to unpack the original distribution file in order to re-do the installation. To remove these files:

```
cd <SDP-home-dir>/bin/$BRAND
/bin/rm -r tmp                  # delete temp files used in bin
cd <SDP-home-dir>/database
/bin/rm de200.dat               # delete ephemeris ASCII file
```

If you plan to use the Ancillary (AA) data access tools, you must now install the AA tools data files, located in an additional compressed tar file, which must be downloaded separately. The installation instructions are located in Section 5.1.4, Installation of AA Tools.

5.1.2.7 Rebuilding the toolkit library

The toolkit installation procedure now makes it easy to rebuild the toolkit library without having to re-install the entire toolkit. This may be useful in the event that any problems are encountered during the installation process.

SCF Installation

To rebuild the toolkit library at an SCF site do the following:

Set directory.

cd <SDP-home-dir>

Type:

bin/INSTALL-Toolkit <install-options> -lib_only

Where, <install-options> are the installation options set in step 2 of Starting the Installation Procedure, above.

SCF Installation

To rebuild the C++ version toolkit library at an SCF site do the following:

Set directory.

cd <SDP-home-dir>

Type:

bin/INSTALL-Toolkit <install-options> -cpp_lib_only

where <install-options> are the installation options set in step 2 of Starting the Installation Procedure, above.

5.1.2.8 NOTES:

1. The SDP Toolkit was built and tested* in a multi–platform environment using the following platforms, operating systems, and compilers:

Table 5-1. SDP Toolkit Development Configuration

Platform	OS	Version	C Compiler	C++ Compiler	FORTRAN
DEC	Digital UNIX	4.0	DEC C 4.10		DEC FORTRAN 4.10
HP	HP–UX	11.0	HP C 11.02.02		HP FORTRAN 11.01.27
IBM	AIX	4.2	IBM C 3.1.4		IBM FORTRAN 3.2.5
SGI	IRIX	6.5	SGI C 7.4.2m	SGI C++	SGI FORTRAN 7.4.2m
Sun	Solaris	5.10	Sun C 5.7	Sun C++ 5.7	Sun FORTRAN 8.1 (f95)
Sun Sparc	Solaris	5.11	gcc		gfortran
Intel Mac 32-bit	Darwin	14.5.0	gcc 4.2.1		gfortran 5.2.0
Intel Mac 64-bit	Darwin	14.5.0	gcc 4.2.1		gfortran 5.2.0
Linux 32-bit	Red Hat Linux 4.4.7-18	2.6.32-358.2.1.e16 .x86_64 #1 SMP	gcc 4.4.7	g++	gfortran 4.4.7
Linux 64-bit	Red Hat Linux 4.4.7-18	Same as above	gcc 4.4.7	g++	gfortran 4.4.7
Windows	7, 10	MS VS .net 2008	Visual c++	Intel Visual Fortran 11.1	
Cygwin	Cygwin	1.7.29	gcc 4.8.2	gfortran 4.8.2	

* Officially SUN, DEC, IBM, HP, Sun5.8, Power Mac, and SGI are not supported anymore and they were not tested for this release

Notes:

a. SGI was also running SGI FORTRAN 90 version 7.0 and NAG FORTRAN-90 2.2.
b. Compilers are provided by platform vendors unless specified.

2. Toolkit architecture type names

   To track architecture dependencies, the toolkit defines the environment variable $BRAND. Following is a list of valid values for this variable, which is referred to throughout this document:

$BRAND	Architecture
dec	DEC Alpha
ibm	IBM AIX
hp	HP 9000, HP-UX11 9000/785
sgi	SGI Power Challenge (old-style 32-bit mode)
sgi32	SGI Power Challenge (new-style 32-bit mode)
sgi64	SGI Power Challenge (64-bit mode)
sun5.8, sun5.9, sun5.10	Sun:SunOS 5.8, OS5.9, OS5.10
linux	LINUX 32-bit Platforms
linux32	64-bit LINUX Platforms for 32-bit mode
linux64	64-bit LINUX Platforms for 64-bit mode
macintel (32-bit, 64-bit)	Macintosh platforms with Intel chip
macintosh	Macintosh Power PC (MAC OS X)
cygwin	CYGWIN 32-bit Platform

3. In order to use the SDP Toolkit libraries and utilities, a number of environment variables MUST be set up to point to SDP directories and files. These variables are automatically set up in User Account Setup section of the installation instructions. They are listed here for reference:

Table 5-2. Required Directory Environment Variables

Name	Value	Description
PGSHOME	<install–path>/TOOLKIT (where <install–path> is the absolute directory path above TOOLKIT)	top level directory
PGSBIN	${PGSHOME}/bin/($BRAND)	executable files
PGSDAT	${PGSHOME}/database/ ($BRAND)	toolkit database files
PGSINC	${PGSHOME}/include	include (header) files
PGSMSG	${PGSHOME}/message	SMF message files
PGSLIB	${PGSHOME}/lib/($BRAND)	library files
PGSOBJ	${PGSHOME}/obj/$BRAND)	toolkit object files
PGSCPPO	${PGSHOME}/objcpp/($BRAND)	toolkit C++ version object files
PGSRUN	${PGSHOME}/runtime	runtime work files
PGSSRC	${PGSHOME}/src	toolkit source files
PGSTST	${PGSHOME}/test	test area
PGS_PC_INFO_FILE	${PGSRUN}/PCF.relB	Process Control File

4. Other toolkit environment variables

   In addition, the makefiles, which are used to build the libraries, require certain machine–specific environment variables. These set compilers, compilation flags and libraries, allowing a single set of makefiles to serve on multiple platforms. The User Account Setup section of the installation instructions explains how to set them up They are listed here for reference:

Table 5-3. Required Compiler and Library Environment Variables

Name	Description
CC	C compiler
CFLAGS	Default C flags (optimize, ANSI)
C_CFH	C w/ cfortran.h callable from FORTRAN
CFHFLAGS	CFLAGS + C_CFH
CPP	C++ compiler
CPPFHFLAGS	Default C++ flags
CPPFHFLAGS	CPPFLAGS
C_F77_CFH	C w/ cfortran.h calling FORTRAN
C_F77_LIB	FORTRAN lib called by C main
F77	FORTRAN compiler
F77FLAGS	Common FORTRAN flags
F77_CFH	FORTRAN callable from C w/ cfortran.h
F77_C_CFH	FORTRAN calling C w/ cfortran.h
CFH_F77	Same as F77_C_CFH
F77_C_LIB	C lib called by FORTRAN main

5. For a complete list of the tools provided with this release of the SDP Toolkit, please refer to Section 1, Table 1–2

6. The majority of the SDP Toolkit functions are written in C. These C–based tools include the file cfortran.h, using it to generate machine–independent FORTRAN bindings.

5.1.3 Compiling User Code with the Toolkit

In order to compile your programs in conjunction with the toolkit, certain flags MUST be set on the compiler command lines. These flags vary, depending on the platform type and operating system.

The toolkit includes command files that set up environment variables to simplify the task of compiling with toolkit code. The user is responsible for ensuring that his or her code complies with the ANSI standards. The following subset is relevant for this discussion:

CC	the name of the C compiler (usually cc)
CFHFLAGS	required C compilation flags (ANSI C mode, optimized)
CPP	the name of the C++ compiler (usually CC)

CFHFLAGS	required C++ compilation flags
F77	FORTRAN compiler (usually f77 in unix, gfortran in Mac/Linux)
F77_CFH	required FORTRAN compilation flags
HDFSYS	a flag used to tell the code what platform is being used
PGSINC	the location of the toolkit include files
PGSLIB	the location of the toolkit library libPGSTK.a
HDFINC	HDF4 include files
HDFLIB	HDF4 Library files
HDF5INC	HDF5 include files
HDF5LIB	HDF5 Library files

To automatically set up these variables for your platform do the following:

for csh users, type:

source <TOOLKIT-HOME-DIRECTORY>/bin/${BRAND}/pgs-dev-env.csh

for ksh users, type:

. <TOOLKIT-HOME-DIRECTORY>/bin/${BRAND}/pgs-dev-env.ksh

where <TOOLKIT-HOME-DIRECTORY> is the location where the toolkit is installed (e.g. /usr/local/PGSTK)

for C++ version, csh users, type:

source <TOOLKIT-HOME-DIRECTORY>/bin/${BRAND}/pgs-dev-env.csh.cpp

for C++ version, ksh users, type:

. <TOOLKIT-HOME-DIRECTORY>/bin/${BRAND}/pgs-dev-env.ksh.cpp

where <TOOLKIT-HOME-DIRECTORY> is the location where the toolkit is installed (e.g. /usr/local/PGSTK)

You may then view the settings of these variables with the command:

$PGSBIN/pgs-flags

NOTE: On some platforms, some of these variables are blank. This is normal—the compile lines given below should work anyway.

You may then view the settings of these variables with the command for the C++ version:

$PGSBIN/pgs-flags-cpp

NOTE: On some platforms, some of these variables are blank. This is normal—the compile lines given below should work anyway.

Once the variables have been set as indicated above, the following command lines may be used as a guide to compiling your programs with the toolkit.

C to object:

```
$CC -c $CFHFLAGS -D$HDFSYS -I$PGSINC myfile.c
```

C++ to object:

```
$CPP -c $CPPFHFLAGS -D$HDFSYS -I$PGSINC myfile.c
```

C to executable:

```
$CC $CFHFLAGS -D$HDFSYS -I$PGSINC -L$PGSLIB \
        myfile.c -lPGSTK (-l...) -o myfile
```

C++ to executable:

```
$CPP $CPPFHFLAGS -D$HDFSYS -I$PGSINC -L$PGSLIB \
        myfile.c -lPGSTK (-l...) -o myfile
```

FORTRAN to object:

```
$F77 -c $F77_CFH myfile.f
```

FORTRAN to executable:

```
$F77 -c $F77_CFH myfile.f $PGSLIB/libPGSTK.a (other libraries ...) \
        -o myfile
```

If the toolkit was built with HDF support included, and your code uses tools that require HDF support, you may use the lines listed below:

C to object:

```
$CC -c $CFHFLAGS -D$HDFSYS -I$PGSINC -I$HDFINC –I$HDF5INC myfile.c
```

C++ to object:

```
$CPP -c $CPPFHFLAGS -D$HDFSYS -I$PGSINC -I$HDFINC –I$HDF5INC \

 myfile.c
```

C to executable:

```
$CC $CFHFLAGS -D$HDFSYS -I$PGSINC -I$HDFINC –I$HDF5INC

-L$PGSLIB -L$HDFLIB –L$HDF5LIB \
        myfile.c -lPGSTK -ldf –lhdf5 (-l ...) -o myfile
```

C++ to executable:

```
$CPP $CPPFHFLAGS -D$HDFSYS -I$PGSINC -I$HDFINC –IHDF5INC

-L$PGSLIB -L$HDFLIB –L$HDF5LIB \
        myfile.c -lPGSTK -ldf  -lhdf5 (-l ...) -o myfile
```

FORTRAN to object:

```
$F77 -c $F77_CFH myfile.f

FORTRAN to executable:
        $F77 -c $F77_CFH myfile.f $PGSLIB/libPGSTK.a $HDFLIB/libdf.a
$HDF5LIB/libhdf5.a \
                (other libraries ...) -o myfile
```

The important thing in this case is that your code gets linked with the HDF4 and HDF5 libraries. You do not need -I$HDFINC or –I$HDF5INC unless your C or C++ code makes direct calls to HDF4 and/or HDF5.

5.1.4 Installation of AA Tools

This section covers installation of the data files needed to use the Ancillary/auxiliary (AA) data access tools. These files include the Digital Chart of the World and other earth sciences data sets. If you do not plan to use these tools or data sets, it is not necessary to install the files.

These files will require approximately 260 Mb of disk space. They may be installed in any location; i.e., they do not have to be stored under the SDP Toolkit home directory.

The tool PGS_AA_dcw MUST have access to the files contained in the four directories named /soamafr, /sasaus, /noamer, /eurnasia in order to work. These files comprise about 80 Mbytes. The other tools (PGS_AA_2/3DRead PGS_AA_2/3Dgeo, PGS_AA_dem) are designed to work with a large range of gridded data sets. Those in the tar file are samples of data from National Geophysical Data Center (NGDC) which need not be maintained by the user; i.e., the user should delete which ever are not pertinent. These files comprise about 180 Mbytes.

The installation script for the AA tools data files is included as part of the main SDP Toolkit distribution. Due to space constraints, the data files themselves are located in a separate compressed tar file, called SDPTK5.1v1.00-AAdata.tar.Z, which must be downloaded separately.

You must first install the SDP Toolkit BEFORE installing the AA tools data files. The AA tools data files installation requires a disk partition with about 400 Mb of free space.

To install the AA tools data files from the tar file:

a. Run the INSTALL-AAdata script

   1. If you have already modified your login files, as in the toolkit installation instructions, simply type:

      INSTALL-AAdata

      from any directory.

   2. If you haven't yet done this, then proceed by typing the following:

      cd <SDP-home-dir>
      bin/INSTALL-AAdata

      where <SDP-home-dir> is the full path of the toolkit home directory.

b. The script contains a default name for the distribution file containing the AA tools data files. That name should be correct for the current release of the toolkit. The script will display the default distribution file name and prompt the user for an override. If the name is correct, press return to continue. If installing from a different distribution file for any reason, please enter the name and press return.

c. By default, the script looks for the tar file in your current directory and also in <SDP-home-dir>. If the file is found in one of the default locations, the script will continue to the next step. Otherwise, please enter the correct location when the script prompts for it.

d. The script then asks where the AAdata directory will be created. The default is <SDP-home-dir>. If you want it installed elsewhere, please enter the pathname when the script prompts for the location. Otherwise, simply hit return to continue.

e. The script asks you to verify the information entered. Type 'y' and hit return to continue. The contents of the distribution file are then extracted into the specified location. Please note that this is a lengthy process that will probably take somewhere between 0.5 and 1.5 hours, depending on your host.

f. The script then asks if the Process Control files, should be patched so that the PRODUCT INPUT FILES directory is set to point to the AA data directory. The default is yes. If you answer no, you must you must edit the Process Control File yourself, in order for the AA tools to work.

g. The script then asks if the distribution file should be removed. The default is no. Once you are satisfied that the files have successfully been installed, you will probably want to get rid of this file, as it takes up a lot of disk space.

   If you wish to get a listing of the files contained in the distribution file, for verification purposes, follow the steps below. Please be aware that this is no small task, as there are literally thousands of data files contained in the distribution file. To see the listing, go to the directory where the distribution file is located and type.

   ```
   zcat SDPTK5.1v1.00-AAdata.tar.Z | tar xvf -
   ```

   You may wish to pipe the output to the UNIX 'more' command, to allow you to see a screen at a time.

   ```
   zcat SDPTK5.1v1.00-AAdata.tar.Z | tar xvf - | more
   ```

This completes the installation of the AA tools data files.

5.2 Instructions on Making Changes to Installation Procedures

The installation procedures given in the previous subsection should work seamlessly for a platform in Table 5–1. This subsection gives instructions on making changes to the installation procedure of subsection 5.1, which may be necessary if one uses a different configuration. Here we give a step–by–step procedure for making these modifications.

In the following procedure, <SDP-home-dir> refers to the SDP Toolkit home directory.

a. After unpacking the tar file, but before running bin/INSTALL, (steps a–e in Section 5.1, corresponding to steps 1–7 in <SDP-home-dir>/README), edit the file INSTALL in <SDP-home-dir>/bin.

   The section starting with the comment at line #266 and ending at line 442 must be modified for your platform. This section consists of a switch block that checks the value

of the environment variable BRAND and sets the flags for each platform accordingly. Modify ONLY the block associated with your platform.

The proper block can be determined from the following table:

Table 5-4. Values of OSTYPE

value of $BRAND	Platform type
sun5.X	Sun Sparc (SunOS 5.X)
hp	HP 9000
dec	DEC Alpha
sgi	SGI Indigo
sgi32	SGI new 32-bit
sgi64	SGI 64-bit
ibm	IBM RS–6000
cray	Cray
linux, linux32, linux64	Linux
cygwin	Cygwin
macintel	MAC with Intel chip (MAC OS X)
macintosh	MAC Power PC (MAC OS X)

Within each block the following variables are set:

Table 5-5. Environment Variables

Name	Description
CC	C compiler
CFLAGS	Default C flags (optimize, ANSI)
C_CFH	C w/ cfortran.h callable from FORTRAN
CFHFLAGS	CFLAGS + C_CFH
CPP	C++ compiler
CPPFLAGS	Default C++ flags
CPPFHFLAGS	CPPFLAGS + CPP_CFH
C_F77_CFH	C w/ cfortran.h calling FORTRAN
C_F77_LIB	FORTRAN lib called by C main
F77	FORTRAN compiler
F77FLAGS	Common FORTRAN flags
F77_CFH	FORTRAN callable from C w/ cfortran.h
F77_C_CFH	FORTRAN calling C w/ cfortran.h
CFH_F77	Same as F77_C_CFH
F77_C_LIB	C lib called by FORTRAN main
HDFSYS	System type as defined by HDF

Modify the code to set these variables to the appropriate values for your compilers. Variables CFHFLAGS, CFH_F77, and HDFSYS should never require modifications. The most important ones are:

CC	the C compiler
CPP	the C++ compiler
F77	the FORTRAN compiler
CFLAGS	MUST set the C compiler for ANSI C code
CPPFLAGS	MUST set the C++ compiler for ANSI C++
F77_CFH	needed when compiling FORTRAN to object code callable from C using cfortran.h
F77_C_CFH	needed when compiling FORTRAN drivers that call C subroutines with FORTRAN bindings written in C using cfortran.h

These flags MUST be properly set in order to build the SDP toolkit.

b. edit the file pgs-dev-env.csh.tmp in <SDP-home-dir>/bin/tmp

The section starting with comment at line #124 and ending at line #445 is identical to the previously mentioned section in the file bin/INSTALL, and must be modified in the same way.

c. continue with the SDP Toolkit installation by running bin/INSTALL (step f in Section 5.1, corresponding to step 6 in <SDP-home-dir>/README).

5.3 Link Instructions

This subsection gives instructions on how to link SDP Toolkit libraries with your code.

The delivery consists of a single SDP Toolkit library called libPGSTK.a.

Here we give generic command lines for linking with this library We use $C_COMPILER, $CPP_COMPILER, and $F77_COMPILER to indicate both the compiler name and any machine–specific compiler flags used by the science software developer. The relevant environment variables must have been previously set up; see the "Installation Procedures" subsection of this section.

To link C code in file "main.c" with the toolkit, on all machines:

```
$C_COMPILER -I$PGSINC -L$PGSLIB main.c -lPGSTK -lm
```

To link C++ code in file "main.c" with the toolkit, on all machines:

```
$CPP_COMPILER -I$PGSINC -L$PGSLIB main.c -lPGSTK -lm
```

To link FORTRAN 77 code in file "main.f" with the toolkit, on all machines:

```
$F77_COMPILER main.f $PGSLIB/libPGSTK.a
```

NOTES:

Specific examples on how to link particular Toolkit functions on the Toolkit development platforms are given with the separately supplied tool test drivers. See the "Test Drivers" in Section 5.4.

If you are using a different development configuration than one of those given in table 5–1 ("SDP Toolkit Development Configuration") of Section 5.1, see Section 5.2 ("Instructions on Making Changes to Installation Procedures") above.

To ensure compatibility of code at the DAACs, science teams are strongly encouraged to use the same compiler switches used by the SDP Toolkit where possible. These switches enforce ANSI/POSIX standards, necessary for compiling the toolkit with the same functionality on all tested platforms; using the same switches in your code makes it more likely that your code will quickly pass integration and test at the DAAC. The compilers and their respective switches are represented by the environment variables **$CC**, **$CFLAGS**, **$CPP, $CPP_FLAGS, $F77**, **$F77FLAGS**, and are defined in the file $PGSHOME/bin/pgs_dev_env.csh and $PGSHOME/bin/pgs_dev_env.csh.cpp respectively. **$CC, $CPP,** and **$F77** contain the names of the C and FORTRAN compilers respectively. **$CFLAGS, CPPFLAGS,** and **$F77** flags contain the compiler switches (options) used by the SDP Toolkit with the C and FORTRAN compilers respectively.

5.4 Test Drivers

Also included with this toolkit delivery is a tar file containing test driver programs.

These test programs are provided to aid the user in the development of software using the toolkit. The user may run the same test cases as included in this file to verify that the toolkit is functioning correctly. These programs were written to support the internal test of the toolkit and are not an official part of the Toolkit delivery; users make use of them at their own risk. No support will be provided to the user of these programs. The tar file contains source code for a driver in C and FORTRAN for each tool; readme files explaining how to use each driver; sample output files; and input files and/or shell scripts, where applicable.

The UNIX command

```
zcat SDPTK5.2.20v1.00_TestDrivers.tar.Z | tar xvf -
```

will create a directory called test drivers beneath the current directory containing all these test files.

5.5 User Feedback Mechanism

The mechanism for handling user feedback, documentation and software discrepancies, and bug reports follows:

a. An account at the ECS Riverdale facility has been set up for user response:

   RVL_PGSTLKIT@raytheon.com

b. Users will e–mail problem reports and comments to the above account. A receipt will be returned to the sender. A work off plan for the discrepancy will be developed and status report issued once a month. Responses will be prioritized based on the severity of the problem and the available resources. Simple bug fixes will be turned around sooner, while requested functional enhancements to the Toolkit will be placed in a recommended requirements database (RRDB) and handled more formally.

c. The following format will be used for email response. It can be found in the tar file in the SDP Release 9 Toolkit 5.2.20 delivery package.

   Name:

   Date:

   EOS Affiliation (DAAC, Instrument, Earth Science Data and Information System (ESDIS), etc.):

   Phone No.:

   Development Environment:

   Computing Platform:

   Operating System:

   Compiler and Compiler Flags:

   Tool Name:

   Problem Description:

   (Please include exact inputs to and outputs from the toolkit call, including error code returned by the function, plus exact error message returned where applicable.)

   Suggested Resolution (include code fixes or workarounds if applicable):

d. A list of Frequently Asked Questions (FAQ) for Toolkits is also available.

   The URL for the SDP Toolkit Frequently Asked Questions (FAQ) page is http://newsroom.gsfc.nasa.gov/sdptoolkit/faq.html

   You can also get there from the EDHS Home Page http://edhs1.gsfc.nasa.gov/. Click on “ECS Development”, then “Toolkit”. The "Toolkit Frequently Asked Questions (FAQ)" link is on the SDP Toolkit webpage.

6. SDP Toolkit Specification

6.1 Introduction

In this section, we give a descriptive list of Toolkit software tools designed to satisfy the requirements listed in *PGS Toolkit Requirements Specification for the ECS Project*, Hughes Information Technology Systems, Inc. 193-801-SD4-001, October 1993 and updated in version through November 1997. The following fields are provided: a name, a synopsis field, a description of each tool, a list of input and output, an error return field, examples, notes, and a cross reference to the target Toolkit requirement(s).

It is assumed that ECS science software requests for system services, for system and resource accesses, file I/O requests, error message transaction, metadata formatting, accesses to spacecraft orbit and attitude, and time and date requests must be made through the Toolkit, as explained in section 4.1. This usage will be tested at integration time at the DAACs. These tools are described in Section 6.2. Other services, such as geographic information data base requests, geolocation tools, scientific and math library calls, requests for physical constants and unit conversions, will be provided; their usage will be encouraged, but not enforced. They are the subject of Section 6.3.

Toolkit routines use the following naming convention:

PGS_GROUPNAME_FUNCTIONALNAME. The GROUPNAME denotes the function of that group of Toolkit routines: IO=Input/Output, SMF=Status/message Facility, MEM=Memory Management, MET=metadata, EPH=Ephemeris/Attitude data access, TD=time and date conversion, PC=ProcessControl, DEM=Digital Elevation Model access, AA=Ancillary Data Access, CBP=Celestial Body Position, GCT=Geo-coordinate Transformation, CUC=Constant and Unit Conversion, CSC=Coordinate System Conversion. The remaining part of the name has sufficient detail to indicate the functionality of the tool. (See also Section 3.2)

There are several C (.h) and FORTRAN (.f) include files listed in the tool descriptions in the following sections, e.g., PGS_IO.h. These files are meant to contain descriptions of data structures, constants; headers; configuration information for data files called by the tools; common symbols; return codes, etc., used in that section. To view these files, look in Toolkit directory $PGSHOME/include.

A note on error handling: Since each function has only one return value; every effort has been made to preserve the most important warning or error value on returning. Given that subordinate functions often have several possible returns, and different users have different priorities, it is always advisable to check the message log in $PGSRUN as well as examining the return. When totally inconsistent behavior is found in a return from a subordinate function, the returned value is PGS_E_TOOLKIT. Example: a Toolkit function passes an internally generated vector, whose length is certain to be nonzero, to a subordinate function. The lower-level function then returns a warning or error return saying that the vector is of zero length; while the higher-level function

returns PGS_E_TOOLKIT. Another example: if a valid spacecraft tag is passed in, but rejected as invalid down the processing line, the error PGS_E_TOOLKIT is returned by the higher-level function. Thus return value PGS_E_TOOLKIT indicates a flaw in the software, the violation of an array boundary, a hardware, compiler, or system error, corrupted data, or some similarly serious condition that invalidates the processing.

6.2 SDP Toolkit Tools-Mandatory

6.2.1 File I/O Tools

This section describes the set of tools used to perform file I/O, including Level 0 access generic and temporary I/O tools, also proposed metadata tools. An explanation of usage of the Toolkit as regards Hierarchical Data Format (HDF) is also included.

6.2.1.1 Level 0 Science Data Access Tools

6.2.1.1.1 Introduction

These Level 0 access tools are used to open and read data from Level 0 data files. These files are generated and formatted by EDOS for AM, PM and AURA platform data, and by the science data processing facility (SDPF) for TRMM platform data.

The Level 0 access tool design has simple user interfaces, and allows science software to do much of the data unpacking in whatever manner is desired. Essentially all header and packet data are returned in character buffers. The packet data is returned a single packet at a time, so the science software can decide whether to store it or to immediately process it.

A complete specification of the Level 0 file formats used in construction of this software is found in Appendix F.

6.2.1.1.2 Design Overview

The design focuses on the idea of a “virtual" data set, consisting of all staged physical L0 files for a particular data type. By data type is meant data that are related in some way; most often this means data with a common application process identifier (APID). There may be many virtual data sets for a given production run. For example, main Clouds and Earth Radiant Energy System (CERES) L0 processing involves science data (APID 54) and housekeeping data (all other APIDs). Each of these two sets of data corresponds to a single virtual data set in the Level 0 tool design. Each virtual data set corresponds to a single logical file ID in the science software and (at the SCF) in the Process Control File (PCF).

For a given run, if a given set of data for a single set of data (science or housekeeping) needs to be broken into more than one file, then each physical file corresponds to a different version of the same logical file ID in the PCF. (This is never expected to be the case for TRMM, but may be for EOS AM (TERRA) or PM (AQUA) or AURA.).

Next is given a brief summary of the functions of the L0 tools. The tools are divided into two groups: one group consisting of required tools for reading L0 data in production software, and one group for use only at the SCF for generation of test data sets.

6.2.1.1.3 Tools for Reading Production L0 Data

PGS_IO_L0_Open sets up internal tables that allow the SDP Toolkit to provide the science software with time-ordered access to file attributes. It opens the first physical file and positions the file pointer at the earliest packet in the staged data. It returns the virtual file handle used by other L0 access tools.

PGS_IO_L0_SetStart is for optionally positioning the virtual file pointer at a start time that is different from the earliest packet in the staged data.

PGS_IO_L0_SetStartCntPkts is for optionally positioning the virtual file pointer at a start time that is different from the earliest packet in the staged data. Also tracks the number of packets skipped in the current file

PGS_IO_L0_GetHeader is for retrieving data from the physical L0 file header; in addition, for TRMM processing, it retrieves data from the file footer, which consists of quality and missing packet information. Data is returned in a simple character buffer.

PGS_IO_L0_GetPacket retrieves a single packet's worth of data. Data is also returned in a simple character buffer by this function.

PGS_IO_L0_Close is for closing a L0 virtual data set.

6.2.1.1.4 Tools for Generating Simple Simulated L0 Data Sets

The above tools satisfy SDP Toolkit requirements for tools that read Level 0 data files; along with these, a means is provided to generate simple simulated Level 0 files. A major portion of TRMM Level 0 processing may be simulated using these files; for EOS AM, PM and AURA platforms, packet and Construction Record File simulation included in the simulator. Provided for simulated file generation are:

L0sim, an executable interactive utility that queries the user about parameters used in creation of a simulated Level 0 data set. It can create file(s) for a single APID, or a housekeeping file with many APIDs; one or many physical files per APID; and many other things. See Appendix E for an example of its use.

PGS_IO_L0_File_Sim, a function callable from C or FORTRAN; it is the underlying function used by *L0sim*. Users who prefer to customize file simulations to fit their own needs may use this function.

6.2.1.1.5 Use of L0 Read Tools In Science Software Processing

Next is presented a brief summary of how science software might use the L0 read tools to do Level 0 processing. A full example of L0 processing using CERES as an example is given in Appendix E. Examples are also provided in individual tool descriptions below.

In the production system, once the required L0 data and other data are staged, the PGE kicks off automatically. During development at the SCF, the developer must first generate file(s) using the simulator tools, then prepare entries in the Process Control File (PCF).

The science code might proceed as follows:

a. Call PGS_IO_L0_Open; with the logical file ID as input parameter used in the PCF. Get back a virtual file handle for use in other tools.

b. Optionally call PGS_PC_GetFileAttr or PGS_PC_GetFileByAttr to read an "attribute" file associated with the L0 data file. For example, for TRMM this might be the detached standard formatted data unit (SFDU) header file.

c. Optionally call PGS_PC_SetStart if a starting time other than the earliest in the data set is desired.

d. Allocate memory for as much data as is desired to save, based on the start and stop times returned from PGS_IO_L0_Open. (In FORTRAN 77 this will have to be hardcoded to some maximum.)

e. While there is still data left, first call PGS_IO_L0_GetHeader to read the physical file header, and also the footer (TRMM quality and accounting capsule (QAC) and missing data unit list (MDUL) data).

f. Call PGS_IO_L0_GetPacket to read a single packet. Repeat until end of data reached, storing the data as desired.

g. If PGS_IO_L0_GetPacket returns a value indicating a new physical file has been opened, loop back to call PGS_IO_L0_GetHeader again to read the new file header.

h. Call PGS_IO_L0_Close to close this virtual data set.

i. If there are more virtual data sets (e.g., APIDs) to process, loop back to call PGS_IO_Gen_Open again.

Note that this algorithm is just one example of how this might be done. Another way is to open several virtual data sets at once.

Please note also that science software is responsible for unpacking headers, packets and footers as it sees fit. Specification of their formats as used in this version of the software appears in Appendix F.

6.2.1.1.6 Special Note on Processing TRMM and ADEOS-II Files

In order to process the Level 0 data files the Level 0 access tools must be able to convert the time found in the data files to TAI. Special preparation is required to do this in the case of TRMM and ADEOS-II.

To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the

Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTCF VALUE>

Where the proper value of the UTCF should be substituted for <UTCF VALUE>.

To properly convert times to or from ADEOS-II s/c clock time the ADEOS-II Time Differential (TMDF) values must be known. These values must be supplied by the user in the Process Control File (PCF). The following lines MUST be contained in the PCF for any process that is converting to or from ADEOS-II s/c clock time:

<UTC VALUE>

10120|ADEOS-II s/c reference time|<S/C REFERENCE TIME>
10121|ADEOS-II ground reference time|<GROUND REFERENCE TIME>
10122|ADEOS-II s/c clock period|<S/C PERIOD>

Where:

the proper value of the S/C clock reference time should be substituted for
< S/C REFERENCE TIME>.

the proper value of the ground reference time should be substituted for
<GROUND REFERENCE TIME> (this time should be in TAI format-see sec. 6.2.7 Time and Date Conversion Tools).

the proper value of the S/C clock period should be substituted for <S/C PERIOD>.

Open a Virtual Data Set

NAME: **PGS_IO_L0_Open**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_Open(
        PGSt_PC_Logical                 file_logical,
        PGSt_tag                        spacecraft_tag,
        PGSt_IO_L0_VirtualDataSet       *virtual_file,
        PGSt_double                     *start_time,
        PGSt_double                     *stop_time)
```

FORTRAN:

```
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 function
PGS_IO_L0_Open(
+       file_logical,
+       spacecraft_tag,
+       virtual_file,
+       start_time,
+       stop_time)

        integer file_logical
        integer spacecraft_tag
        integer virtual_file
        double precision start_time
        double precision stop_time
```

DESCRIPTION

This tool opens the virtual data set pointed to by file_logical. A virtual Level 0 data set is defined by the set of physical data files that have been staged for this Level 0 process.

The tool returns a descriptor that is used by all the Level 0 tools to access the specified virtual data set. The tool also returns the start and stop times of this virtual data set.

INPUTS: file_logical-The logical file descriptor for this virtual data set, as given in the Process Control File

spacecraft_tag-The tag identifying which of the supported spacecraft platforms generated this virtual data set. Must be either PGSd_EOS_AM, PGSd_EOS_AURA, PGSd_EOS_PM_GIIS, PGSd_EOS_PM_GIRD, PGSd_TRMM, or PGSd_ADEOS_II.

OUTPUTS: virtual_file-The file descriptor used by all other Level 0 access tools to refer to the virtual data set

start_time-The start time of this virtual data set

stop_time-The stop time of this virtual data set

Time format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993

RETURNS:

Table 6-1. PGS_IO_L0_Open Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_W_L0_CORRUPT_FILE_HDR	Corrupted file header
PGSIO_E_L0_BAD_SPACECRAFT_TAG	Invalid spacecraft tag
PGSIO_E_L0_INIT_FILE_TABLE	Error during read of physical file header for initialization
PGSIO_E_L0_INVALID_FILE_LOGICAL	Failed to process this file logical in process control file
PGSIO_E_L0_MAP_VERSIONS	Failed to initialize internal physical file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_SEEK_1ST_PACKET	Can't find 1st packet in dataset

EXAMPLES: Prepare in part for Lightening Imaging Sensor (LIS) Level 0 processing by opening the LIS/TRMM Level 0 virtual data set for science APID 61.

For TRMM, there is expected to be only one physical file per APID per day. In this case each virtual data set (APID) corresponds to exactly one physical file.

At the SCF, you must prepare entries of the following form in the Process Control File:

```
?  PRODUCT INPUT FILES
# [ set env var PGS_PRODUCT_INPUT for default location ]
#
61|TRMM_G0091_1997-11-
    01T00:00:00Z_dataset_V01_01|||TRMM_G0091_1997-11-
    01T00:00:00Z_sfdu_V01_01|1
```

(Here the logical ID used is arbitrarily set to the APID.)

Note: In the above Process Control File entry, 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. Use functions PGS_IO_PC_GetFileAttr or PGS_IO_PC_GetFileByAttr to retrieve data from this file. Also, the PCF entry must appear on a single line, and not be broken into several lines as shown here.

C:

```
#define SCIENCE_FILE 61

PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_PC_Logical               file_logical;
PGSt_tag                      spacecraft_tag;
PGSt_double                   start_time;
PGSt_double                   stop_time;
PGSt_SMF_status               returnStatus;

file_logical = SCIENCE_FILE;
spacecraft_tag = PGSd_TRMM;

returnStatus = PGS_IO_L0_Open(
      file_logical,
      spacecraft_tag,
      &virtual_file,
      &start_time,
      &stop_time);

/#     Virtual file handle virtual_file may now be used as
input to other L0 access tools #/
```

FORTRAN:

```
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=61)
integer           pgs_io_l0_open
integer           file_logical
integer           spacecraft_tag
integer           virtual_file
double precision  start_time
```

```
double precision  stop_time
integer           returnstatus

file_logical = SCIENCE_FILE
spacecraft_tag = PGSd_TRMM

returnstatus = pgs_io_l0_open(
      file_logical,
      spacecraft_tag,
      virtual_file,
      start_time,
      stop_time)

C     Virtual file handle virtual_file may now be used as input to

C     other L0 access tools
```

NOTES: A virtual data set is defined by a set of one or more related Level 0 physical files. For example, it might consist of all physical files corresponding to a single TRMM science application ID (APID) for a single production run. In the case of EDOS formatted Level 0 data files, a virtual data set consists of all physical files comprising an EDOS PDS/EDS. Only one PDS/EDS is allowed per virtual file.

The maximum number of virtual data sets that may be open at any one time is 20.

This function must be called first; before any other Toolkit Level 0 access tools are called.

A virtual data set may consist of several physical files. In this case the files are listed in the process control file with the same logical ID (1st field) but different instance number (last field).

The physical file version corresponding to the first time-ordered set of packets for the virtual data set is opened by this tool. The file pointer is left positioned so that the next call to PGS_IO_L0_GetPacket will read the first packet in the file.

To get file header and footer (TRMM only) information for the newly opened physical file, use tool PGS_IO_L0_GetHeader. A rudimentary check is done on the header of the first physical file of the virtual data set. If an error is found in the header this function will return the value PGSIO_W_L0_CORRUPT_HEADER. The file will be opened anyway and the user may use the function PGS_IO_L0_GetHeader() to retrieve the header. That function will give a more detailed analysis of the problem. Users should be aware, though, that if they proceed after getting the return PGSIO_W_L0_CORRUPT_HEADER from this function they do so at THEIR OWN RISK. This return value indicates that the file header is

corrupt and the use of any further Toolkit functions to attempt to read the file may produce unexpected results.

In the case of EDOS formatted Level 0 data files (PDS/EDS) the “header” returned will actually be the Construction Record.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

Note Regarding Use of the Process Control File:

If more than one physical file is associated with a given virtual data set, the entries in the Process Control File that map the data set from file_logical to the physical files must appear in reverse numerical order. For example, in a three-file data set, file instance #3 is listed first and file instance #1 is listed last. This mechanism will become transparent in the production system.

REQUIREMENTS: PGSTK-0140, PGSTK-0190, PGSTK-0240

Set Start Time

NAME: **PGS_IO_L0_SetStart**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_SetStart(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_double                     start_time)
```

FORTRAN:

```
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 function PGS_IO_L0_SetStart(virtual_file, start_time)
        integer             virtual_file
        double precision    start_time
```

DESCRIPTION Sets the virtual file pointer so that the next call to the tool PGS_IO_L0_GetPacket will read the first available packet at or after the specified time.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

start_time-The start time of the desired packet. Format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993.

OUTPUTS: NONE

RETURNS:

Table 6-2. PGS_IO_L0_SetStart Returns (1 of 2)

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_W_L0_TIME_NOT_FOUND	Requested start time not found; file pointer position was unchanged
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file

Table 6-2. PGS_IO_L0_SetStart Returns (2 of 2)

Return	Description
PGSIO_E_L0_SEEK_PACKET	Unable to find requested packet
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed
PGSIO_W_L0_BITFLIP_IN_MICSEC	Bit flip problem in the micro second field of a packet time

EXAMPLES: Set the time to start processing at 20 minutes after the data set start time. Examples assume the data set start time has previously been returned from PGS_IO_L0_Open.

C:

```
PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_double                   start_time;
PGSt_double                   new_start_time;
PGSt_SMF_status               returnStatus;

new_start_time = start_time + 1200.0;

returnStatus = PGS_IO_L0_SetStart( virtual_file,
      new_start_time);
if ((returnStatus != PGS_S_SUCCESS)&& (returnStatus
!=PGSIO_W_L0_BITFLIP_IN_MICSEC))
{
    goto EXCEPTION;  /# GO TO EXCEPTION HANDLING #/
}

else

      do something else;

}
```

FORTRAN:

```
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)
```

```
if (returnStatus .ne.
PGS_S_SUCCESS.and.returnStatus.ne.PGSIO_W_L0_BITFLIP_IN_MICS
EC) goto EXCEPTION
```

NOTES: Normal return is PGS_S_SUCCESS. During the search for the desired packet for AM spacecraft a packet with bitflip problem in the micro second field may be encountered. In that case the problematic packet will be ignored and the search will continue. If no other errors occur them the tool will return PGSIO_W_L0_BITFLIP_IN_MICSEC.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

RELEASE NOTES:

There are no Release Notes.

REQUIREMENTS: PGSTK-0140, PGSTK-0200, PGSTK-0220, PGSTK-0240

Set Start Time and Count Packets

NAME: **PGS_IO_L0_SetStartCntPkts**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_SetStart(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_double                     start_time
        PGSt_integer*                   totpacket_skip)
```

FORTRAN:

```
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 function PGS_IO_L0_SetStart(virtual_file, start_time,
                                    totpacket_skip)
        integer             virtual_file
        double precision    start_time
        integer             totpacket_skip
```

DESCRIPTION Sets the virtual file pointer so that the next call to the tool PGS_IO_L0_GetPacket will read the first available packet at or after the specified time. Also tracks the number of packets skipped in the current file.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

start_time-The start time of the desired packet. Format is TAI: continuous seconds since 12AM UTC Jan. 1, 1993.

OUTPUTS: totpacket_skip – The total number of packets skipped before the desired packet selected at the specified time

RETURNS:

Table 6-3. PGS_IO_L0_SetStart Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_W_L0_TIME_NOT_FOUND	Requested start time not found; file pointer position was unchanged
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSIO_E_L0_SEEK_PACKET	Unable to find requested packet
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed
PGSIO_W_L0_BITFLIP_IN_MICSEC	Bit flip problem in the micro second field of a packet time

EXAMPLES: Set the time to start processing at 20 minutes after the data set start time. Examples assume the data set start time has previously been returned from PGS_IO_L0_Open.

C:

```
PGSt_IO_L0_VirtualDataSet     virtual_file;
PGSt_double                   start_time;
PGSt_double                   new_start_time;
PGSt_SMF_status               returnStatus;
PGSt_integer                  totalpacket_skip;

new_start_time = start_time + 1200.0;

returnStatus = PGS_IO_L0_SetStart( virtual_file,
      new_start_time, &totalpacket_skip);
if ((returnStatus != PGS_S_SUCCESS)&&(returnStatus
!=PSGIO_W_L0_BITFLIP_IN_MICSEC))
{
    goto EXCEPTION;  /# GO TO EXCEPTION HANDLING #/
}

else

}

      do something else;

}
```

FORTRAN:

```
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
integer           totalpacket_skip
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,totalpacket_skip)
if (returnStatus .ne.
PGS_S_SUCCESS.and.returnStatus.ne.PGSIO_W_L0_BITFLIP_IN_MICS
EC) goto EXCEPTION
```

NOTES: Normal return is PGS_S_SUCCESS. During the search for the desired packet for AM spacecraft a packet with bit flip problem in the micro second field may be encountered. In that case the problematic packet will be ignored and the search will continue. If no other errors occur then the tool will return PGSIO_W_L0_BITFLIP_IN_MICSEC.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

RELEASE NOTES:

There are no Release Notes.

REQUIREMENTS: PGSTK-0140, PGSTK-0200, PGSTK-0220, PGSTK-0240

Get Header Data

NAME: **PGS_IO_L0_GetHeader**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_GetHeader(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_integer                    header_buffer_size,
        PGSt_IO_L0_Header               *header_buffer,
        PGSt_integer                    footer_buffer_size,
        PGSt_IO_L0_Footer               *footer_buffer)
```

FORTRAN:

```
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 function PGS_IO_L0_GetHeader(virtual_file, header_buffer_size,
                                     header_buffer,
                                     footer_buffer_size,
                                     footer_buffer)
        integer virtual_file
        integer header_buffer_size
        character*(*)   header_buffer
        integer footer_buffer_size
        character*(*)   footer_buffer
```

DESCRIPTION: This tool reads header and footer information for the currently open physical file into the user-supplied buffers. It is intended to be called whenever the file header and footer data change, though it may be called at any time. In the case EDOS formatted files this tool will return the entire contents of the PDS/EDS Construction Record.

The file header and footer data will change whenever a call to one of the tools causes a new physical file to be opened. This will always occur upon a call to PGS_IO_L0_Open, and may also occur upon calls to PGS_IO_L0_SetStart and PGS_IO_L0_GetPacket. These latter two signal this event via a return status code of PGSIO_M_L0_HEADER_CHANGED. In the case of EDOS files, which

have no headers, no notice will be given when a new physical file is opened. Typical use of this tool is in a loop of calls to read data packets.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open

header_buffer_size-Size in bytes of user-supplied header buffer

footer_buffer_size-Size in bytes of user-supplied footer data buffer. If 0, do not read footer data (TRMM only)

OUTPUTS: header_buffer-User-supplied buffer containing the header, read in from the current physical file

footer_buffer-User-supplied buffer containing the footer data, read in from the current physical file (TRMM only)

RETURNS:

Table 6-4. PGS_IO_L0_GetHeader Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_BAD_BUF_SIZ	Buffer size must be a positive integer
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_E_L0_FSEEK	Failed to locate requested byte in file
PGSIO_W_L0_HDR_TIME_ORDER	Time of last packet is earlier than first packet in file header
PGSIO_E_L0_BAD_VAR_HDR_SIZE	Size of the variable header is invalid
PGSIO_W_L0_BAD_PKT_DATA_SIZE	Total size of packet data is invalid
PGSIO_W_L0_BAD_PACKET_COUNT	Total number of packets is invalid
PGSIO_W_L0_BAD_FOOTER_SIZE	Size of the file footer is invalid
PGSIO_W_L0_ZERO_PACKET_COUNT	Total number of packets is zero
PGSIO_W_L0_HDR_BUF_TRUNCATE	Insufficient header buffer size - data
PGSIO_W_L0_FTR_BUF_TRUNCATE	Insufficient footer buffer size - data
PGSIO_W_L0_ALL_BUF_TRUNCATE	Insufficient header buffer AND footer buffer sizes - data truncated
PGSIO_E_L0_UNEXPECTED_EOF	Encountered unexpected end-of-file
PGS_E_UNIX	UNIX error (check log file for type of error)
PGSIO_E_L0_BAD_SPACECRAFT_TAG	Invalid spacecraft tag

EXAMPLES: The example shows how to use this function in conjunction with PGS_IO_L0_GetPacket to read Level 0 data from a single virtual data set. This algorithm works whether the virtual data set consists of only one, or of several physical files. All data in the virtual data set are read.

For clarity, error handling is omitted from the examples.

C:

```
#define HEADER_BUFFER_MAX    556  /# max # header bytes #/
#define FOOTER_BUFFER_MAX 100000  /# max # footer bytes #/
#define PACKET_BUFFER_MAX   7132  /# max # packet bytes #/

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_IO_L0_Packet       packet_buf[PACKET_BUFFER_MAX];

PGSt_integer file_loop_flag;
PGSt_integer packet_loop_flag;

file_loop_flag = 1;
while( file_loop_flag )
{
   returnStatus = PGS_IO_L0_GetHeader( virtual_file,
                   HEADER_BUFFER_MAX, header_buffer,
                   FOOTER_BUFFER_MAX, footer_buffer );

/#   Unpack and/or save or process header and footer data
      here #/

   packet_loop_flag = 1;
   while( packet_loop_flag )
   {
      returnStatus = PGS_IO_L0_GetPacket(
                  virtual_file, PACKET_BUFFER_MAX,
                  packet_buf );

      switch (returnStatus)
      {
         case PGSIO_M_L0_HEADER_CHANGED:
         /# end of this physical file #/
            packet_loop_flag = 0;
            break;

         case PGSIO_W_L0_END_OF_VIRTUAL_DS:
        /# end of this virtual data set #/
            file_loop_flag = 0;
            packet_loop_flag = 0;
            break;
       }

/#   Unpack and/or save or process packet data here #/

   }     /# End while (packet_Loop_flag) #/

}    /# End while (file_Loop_flag) #/
```

FORTRAN:

```
        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'

        character*556     header_buffer
        character*7132    packet_buffer
        character*100000  footer_buffer
        integer           pgs_io_l0_getheader
        integer           pgs_io_l0_getpacket
        integer           virtual_file
        integer           file_loop_flag
        integer           packet_loop_flag
        integer           returnstatus

        file_loop_flag = 1
        do while( file_loop_flag )

           returnstatus = pgs_io_l0_getheader( virtual_file,
                556, header_buffer,
                100000, footer_buffer )

C   Unpack and/or save or process header and footer data here

        packet_loop_flag = 1

        do while( packet_loop_flag )

        returnStatus = pgs_io_l0_getpacket(

        virtual_file, PACKET_BUFFER_MAX, packet_buf )

        if (returnstatus .eq. PGSIO_M_L0_HEADER_CHANGED) then

C   end of this physical file

        packet_loop_flag = 0

        else if (returnstatus .eq.
        PGSIO_W_L0_END_OF_VIRTUAL_DS) then

C    end of this virtual data set
        file_loop_flag = 0
        packet_loop_flag = 0
        end if

C   Unpack and/or save or process packet data here
```

```
end do
end do
```

NOTES: Memory must be allocated to the output buffers before this tool is called. Failure to do this may result in a core dump. (In FORTRAN 77, the buffer CHARACTER array length must be hardcoded.)

If the tool determines that the actual size of the file header or footer is larger than the user-supplied buffer size, the header or footer data is truncated to fit the user buffer. In this case, the return status will be PGSIO_W_L0_HDR_BUF_TRUNCATE (if header buffer too small), PGSIO_W_L0_FTR_BUF_TRUNCATE (if footer buffer too small), or .PGSIO_W_L0_ALL_BUF_TRUNCATE (if both buffers too small).

To retrieve the header and footer information from the first physical file in a virtual data set, this tool must be called after first having opened the virtual data set using the tool PGS_IO_L0_Open. To retrieve the header and footer information from subsequent physical files within a virtual data set, this tool should be called after the science software receives the return status PGSIO_M_L0_HEADER_CHANGED from the tool PGS_IO_L0_GetPacket.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called. If the header of the currently open physical file is found to be corrupted, this function will return a warning to that effect:

PGSIO_W_L0_HDR_TIME_ORDER
PGSIO_E_L0_BAD_VAR_HDR_SIZE
PGSIO_W_L0_BAD_PKT_DATA_SIZE
PGSIO_W_L0_BAD_PACKET_COUNT
PGSIO_W_L0_BAD_FOOTER_SIZE
PGSIO_W_L0_ZERO_PACKET_COUNT

The above returns indicate an error was found in the file header. The header buffer will be returned, although it MAY be truncated. Similarly the footer buffer (TRMM only) may be truncated or even missing if the corrupt header file indicated that the start of the footer buffer was at an offset (in the file) greater than the size of the physical file. The user is cautioned to check the returned buffer(s) carefully in these cases. Further, the user is cautioned that while the function PGS_IO_L0_GetPacket() may still be called, that function may produce unexpected results if the file header is corrupt.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

REQUIREMENTS: PGSTK-0140, PGSTK-0210, PGSTK-0230, PGSTK-0240

Get a Single Packet

NAME: **PGS_IO_L0_GetPacket**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_GetPacket(
        PGSt_IO_L0_VirtualDataSet       virtual_file,
        PGSt_integer                    packet_buffer_size,
        PGSt_IO_L0_Packet               *packet_buffer)
```

FORTRAN:

```
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 function PGS_IO_L0_GetPacket(virtual_file, packet_buffer_size,
                                     packet_buffer)
    integer virtual_file
    integer packet_buffer_size
    character*(*)   packet_buffer
```

DESCRIPTION: Reads a single data packet from a Level 0 virtual data set into the user-supplied buffer.

INPUTS: virtual_file-The file descriptor for this virtual data set returned by PGS_IO_L0_Open.

packet_buffer_size-Size in bytes of user-supplied packet buffer.

OUTPUTS: packet_buffer-User-supplied buffer containing the data packet read in from the specified virtual data set.

RETURNS:

Table 6-5. PGS_IO_L0_GetPacket Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_E_L0_PHYSICAL_NOT_OPEN	No physical file currently open for this virtual data set
PGSIO_E_L0_PKT_BUF_OVERFLOW	Packet buffer too small; no data was read
PGSIO_E_L0_UNEXPECTED_EOF	Encountered unexpected end-of-file
PGSIO_W_L0_PKT_BUF_TRUNCATE	Insufficient buffer size-data truncated
PGSIO_W_L0_END_OF_VIRTUAL_DS	Reached end of the current data set
PGSIO_M_L0_HEADER_CHANGED	New physical file open-file header has changed
PGSIO_E_L0_NEXT_PHYSICAL	Error opening next physical file in sequence
PGSIO_E_L0_SEEK_1ST_PACKET	Can't find first packet in dataset
PGSIO_W_L0_BUFTRUNC_END_DS	Insufficient packet buffer size-reached end of the current data set
PGSIO_W_L0_BUFTRUNC_HDR_CHG	Insufficient packet buffer size-new physical file open-file header has changed
PGSIO_E_L0_BUFTRUNC_NXTFILE	Insufficient buffer size-error opening next physical file in sequence
PGS_E_UNIX	UNIX error (check StatusLog file)

EXAMPLES: The example shows how to use this function in conjunction with PGS_IO_L0_GetPacket to read Level 0 data from a single virtual data set. This algorithm works whether the virtual data set consists of only one, or of several physical files. All data in the virtual data set are read.

For clarity, error handling is omitted from the examples.

C:

```
#define HEADER_BUFFER_MAX    556  /# max # header bytes #/
#define FOOTER_BUFFER_MAX 100000  /# max # footer bytes #/
#define PACKET_BUFFER_MAX   7132  /# max # packet bytes #/

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_IO_L0_Packet       packet_buf[PACKET_BUFFER_MAX];

PGSt_integer file_loop_flag;
PGSt_integer packet_loop_flag;

file_loop_flag = 1;
while( file_loop_flag )
{
   returnStatus = PGS_IO_L0_GetHeader( virtual_file,
```

```
                        HEADER_BUFFER_MAX, header_buffer,
                        FOOTER_BUFFER_MAX, footer_buffer );

/#   Unpack and/or save or process header and footer data
      here #/

   packet_loop_flag = 1;
   while( packet_loop_flag )
   {
      returnStatus = PGS_IO_L0_GetPacket(
                   virtual_file, PACKET_BUFFER_MAX,
                   packet_buf );

      switch (returnStatus)
      {
         case PGSIO_M_L0_HEADER_CHANGED:
         /# end of this physical file #/
            packet_loop_flag = 0;
            break;

         case PGSIO_W_L0_END_OF_VIRTUAL_DS:
        /# end of this virtual data set #/
           file_loop_flag = 0;
           packet_loop_flag = 0;
           break;
       }

/#   Unpack and/or save or process packet data here #/

   }     /# End while (packet_loop_flag) #/

}    /# End while (file_loop_flag) #/
```

FORTRAN:

```
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'

character*556     header_buffer
character*7132    packet_buffer
character*100000  footer_buffer
integer           pgs_io_l0_getheader
integer           pgs_io_l0_getpacket
integer           virtual_file
integer           file_loop_flag
```

```
                integer           packet_loop_flag
                integer     returnstatus

                file_loop_flag = 1
                do while( file_loop_flag )

                   returnstatus = pgs_io_l0_getheader( virtual_file,
                         556, header_buffer,
                         100000, footer_buffer )

C   Unpack and/or save or process header and footer data here

                packet_loop_flag = 1

                do while( packet_loop_flag )

                returnStatus = pgs_io_l0_getpacket(

                virtual_file, PACKET_BUFFER_MAX, packet_buf )

                if (returnstatus .eq. PGSIO_M_L0_HEADER_CHANGED) then

C   end of this physical file

                packet_loop_flag = 0

                else if (returnstatus .eq.
                PGSIO_W_L0_END_OF_VIRTUAL_DS) then

C    end of this virtual data set

                file_loop_flag = 0

                packet_loop_flag = 0

                end if

C   Unpack and/or save or process packet data here

                end do

                end do
```

NOTES: Memory must be allocated to the output buffer before this tool is called. Failure to do this may result in a core dump. (In FORTRAN 77, the buffer CHARACTER array length must be hardcoded.)

Normal return is PGS_S_SUCCESS. If getting the next packet requires that a new physical file be opened, the header and quality data will change. In this case, the return status is set to PGSIO_M_L0_HEADER_CHANGED. This allows the user to test the return status and get updated header and quality data using the tool

PGS_IO_L0_GetHeader, in the case where there is more than one physical file per virtual data set.

If the tool determines that the size of the packet is larger than the user buffer size, as specified by the parameter packet_size, it will truncate the packet to fit the user buffer. In this case, the return status will be PGSIO_W_L0_BUFFER_TRUNCATE.

Packet formats for TRMM, EOS AM (GIIS), EOS PM (GIRD and GIIS) and EOS AURA (GIRD) are supported.

The source document for EOS AM, EOS PM and EOS AURA packet data format is the Interface Control Document Between The Earth Observing System (EOS) Data and Operation System (EDOS) and the EOS Ground System (EGS) Elements (510-ICD-EDOS/EGS CDPL B301), Mission Operations and Data System Directorate, Goddard Space Flight Center, November 5, 1999.

A virtual data set must have been opened by PGS_IO_L0_Open before this function is called.

This function returns no data if the packet buffer size is less than 6 bytes (the primary packet header size). It returns a warning and a truncated buffer if the packet buffer size is more than 6 bytes but less than the actual packet length.

REQUIREMENTS: PGSTK-0140, PGSTK-0200, 0240

Close a Virtual Data Set

NAME: **PGS_IO_L0_Close**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_L0_Close(
        PGSt_IO_L0_VirtualDataSet virtual_file)
```

FORTRAN:

```
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 function PGS_IO_L0_Close(virtual_file)
        integer virtual_file
```

DESCRIPTION: This tool closes a virtual data set opened by a call to the tool PGS_IO_L0_Open.

INPUTS: virtual_file-The file descriptor for this virtual data set, returned by the call to PGS_IO_L0_Open.

OUTPUTS: NONE

RETURNS:

Table 6-6. PGS_IO_L0_Close Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_VIRTUAL_DS_NOT_OPEN	Virtual data set is not open
PGSIO_E_L0_MANAGE_TABLE	Error accessing internal virtual file table
PGSIO_W_L0_PHYSICAL_CLOSE	Failed to close physical file

EXAMPLES: Close a virtual data set opened with a call to PGS_IO_L0_Open. Go to exception handling if there was an error.

C:

```
PGSt_SMF_status returnStatus = PGS_S_SUCCESS;
PGSt_IO_L0_VirtualDataSet  virtual_file;

returnStatus = PGS_IO_L0_Close(virtual_file);
if (returnStatus != PGS_S_SUCCESS) goto EXCEPTION;
```

FORTRAN:

```
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     returnstatus
integer     virtual_file

returnstatus = pgs_io_l0_close(virtual_file)
if (returnstatus != PGS_S_SUCCESS) goto 9999
```

NOTES: If a physical file is currently open, PGS_IO_Gen_Close is called to close it. Otherwise this step is skipped. In either case, the return will be PGS_S_SUCCESS.

REQUIREMENTS: PGSTK-0140, PGSTK-0190

Create a Simulated Level 0 Data File

NAME: **PGS_IO_L0_File_Sim**

SYNOPSIS:

C:

```
#include <PGS_IO.h>
#include <PGS_IO_L0.h>

PGSt_SMF_status
PGS_IO_L0_File_Sim(
        PGSt_tag            spacecraftTag,
        PGSt_integer        appID[],
        PGSt_integer        firstPacketNum
        char                startUTC[28],
        PGSt_integer        numValues,
        PGSt_double         timeInterval,
        PGSt_integer        dataLength[],
        PGSt_integer        otherFlags[2],
        char                *filename,
        void                *appData,
        PGSt_uinteger       qualMissLen[2])
        void                *qualData)
        void                *missData)
```

FORTRAN:

```
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 function pgs_io_l0_file_sim ( spacecrafttag, appid,firstpacketnum,
                                      startutc, numvalues,
                                      timeinterval, datalength,
                                      otherflags, filename,appdata,
                                      qualmisslen, qualdata,
                                      missdata )
        integer             spacecrafttag
        integer             appid(*)
        integer             firstpacketnum
        character*27        startutc
        integer             numvalues
        double precision    timeinterval
        integer             datalength(*)
```

integer	otherflags(2)
character*(*)	filename
(any)	appdata
integer	qualmisslen(2)
(any)	qualdata
(any)	missdata

DESCRIPTION: This tool creates file(s) containing simulated Level 0 data, each of which has a file header, packet data, and a file footer. For TRMM, a detached SFDU header file is also created for each Level 0 data file.

INPUTS: spacecraftTag-The spacecraft identifier desired for the output data.

appID-Array of application process identifiers (APIDs), one for each packet to be generated

firstPacketNum-Value of Packet Sequence Count to use for the initial packet

startUTC-The UTC time of the first packet. Formats supported:

a) YYYY-MM-DDThh:mm:ss.dddddd
b) YYYY-DDDThh:mm:ss.dddddd

numValues-The number of packets to generate

timeInterval-Time interval (in seconds) between packets

dataLength-Array of lengths, in bytes, of the Application Data for each packet. Does not include lengths of primary and secondary packet headers.

otherFlags-Array of length 2 with file header values

otherFlags[0]: bit-packed “Processing Options” byte TRMM values:

bit 3 on-Redundant Data Deleted
bit 6 on-Data Merging
bit 7 on-RS Decoding
bits 1,2,4,5,8-always off

For example, to simulate Redundant Data Deleted and RS Decoding, turn bits 3 and 7 on, which is decimal 68.

So set otherFlags[0]=68.

otherFlags[1]: “Data type Flags” byte TRMM values:

otherFlags[1]=1, Routine production data
otherFlags[1]=2, Quicklook data

(NOTE: These two fields are simply written to the appropriate place in the file header; no processing is done in this function based on their values.)

filename-The name of the file to be created containing the L0 packets.

appData-Optional user-defined input of the packet application data field. Does not include packet header data.

In C, if appData=NULL, a block of data of length equal to the largest value in array dataLength is filled with zeroes, for each packet.

(The remaining inputs are for TRMM file footer processing only. They are ignored for other platforms.)

qualMissLen-Array of length 2 with file footer section lengths qualMissLen[0]: quality (QAC) buffer length if qualMissLen[0]=0, no quality data are written to the file qualMissLen[1]: missing data (MDUL) buffer length if qualMissLen[1]=0 or qualMissLen[0]=0, no missing data are written to the file (QAC length and MDUL length are always written to the file)

qualData-Quality and Accounting Capsule (QAC) data In C, if qualData=NULL, a block of data of length qualMissLen[0] is filled with zeroes and written to the file. (In FORTRAN you pass a zero-filled array for this.)

missData-Missing Data Unit List (MDUL) data In C, if missData=NULL, a block of data of length qualMissLen[1] is filled with zeroes and written to the file. (In FORTRAN you pass a zero-filled array for this.)

OUTPUTS: NONE

RETURNS:

Table 6-7. PGS_IO_L0_File_Sim Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_L0_BAD_NUM_PKTS	Illegal number of packets
PGSIO_E_L0_BAD_APP_ID	At least 1 packet had a bad Application ID
PGSIO_E_L0_BAD_FIRST_PKTNUM	Illegal first packet number
PGSTD_E_SC_TAG_UNKNOWN	spacecraft tag is unknown or not currently supported
PGSIO_E_L0_BAD_DATA_LENGTH	At least 1 packet had a bad data length
PGSIO_E_L0_BAD_NUM_APP_IDS	Illegal number of differing Application IDs
PGSTD_E_TIME_FMT_ERROR	Error in ASCII time string format (generic format: YYYY-MM-DDThh:mm:ss.ddddddZ)
PGSTD_E_TIME_VALUE_ERROR	Error in ASCII time string value (e.g., hours > 23)
PGS_E_TOOLKIT	Unspecified Toolkit error (check StatusLog file)
PGS_E_UNIX	UNIX error (check StatusLog file)
PGSMEM_E_MAXSIZE	Maximum memory size reached: %d in bytes
PGSIO_E_L0_PHYSICAL_OPEN	Unable to open physical file
PGSTD_E_DATE_OUT_OF_RANGE	the input time is outside the range of allowable values for the spacecraft clock

EXAMPLES: Generate a CERES L0 science telemetry file named TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01, containing 3 packets of different lengths, starting at midnight Dec. 1, 1997 and spaced at 6.6 second intervals; also add QAC and MDUL data, filled with zeroes.

C:

```
#define N 3

PGSt_tag     spacecraftTag = TRMM;
PGSt_integer appID[N] = {54,54,54};
PGSt_integer firstPacketNum = 1;
char         *startUTC = "1997-12-01T00:00:00";
PGSt_integer numValues = N;
PGSt_double  timeInterval = 6.6;
PGSt_integer dataLength[N];
PGSt_integer otherFlags[2];
char         *filename
        = "TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01";
char         appData[9000];
PGSt_uinteger qualMissLen[2]={28,16};
char         *qualData=NULL;
char         *missData=NULL;

PGSt_SMF_status returnStatus;

otherFlags[0] = 68; /* Redundant Data Deleted & RS Decoding
                       */
otherFlags[1] =  1; /* Routine production data */

/* Set lengths of packet application data */
dataLength[0] = 2000;
dataLength[1] = 3000;
dataLength[2] = 4000;

/* Fill appData buffer as desired here.

Do not include packet header data—it is filled by this
      tool.

Fill first 2000 bytes with first packet data,

next 3000 bytes with second packet data,

last 4000 bytes with third packet data */

/* Create simulated file */

returnStatus =
  PGS_IO_L0_File_Sim(
      spacecraftTag,
```

```
                    appID,
                    firstPacketNum,
                    startUTC,
                    numValues,
                    timeInterval,
                    dataLength,
                    otherFlags,
                    filename,
                    appData,
                    qualMissLen,
                    qualData,
                    missData,
                    );
```

FORTRAN:

```
implicit none

integer pgs_io_l0_file_sim

integer spacecraftTag
integer appid(3)
integer firstpacketnum
character*27 startutc
integer numvalues
double precision timeinterval
integer datalength(3)
integer otherflags(2)
character*256 filename
character*9000 appdata
integer qualmisslen(2)
character*28 qualdata
character*16 missdata

integer returnstatus

spacecraftTag = TRMM
appid(1) = 54
appid(2) = 54
appid(3) = 54
firstpacketnum = 1
startutc = '1994-12-31T12:00:00.000000'
numvalues = 3
timeinterval = 6.6

C   Set lengths of packet application data
datalength(1) = 2000
datalength(2) = 3000
datalength(3) = 4000
```

```
C   Fill data to write to file header
                otherflags(1) = 68 ! Redundant Data Deleted & RS Decoding
                otherflags(2) =  1 ! Routine production data
                filename = 'TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01'
                qualmisslen(1) = 28
                qualmisslen(2) = 16

C   Fill appData buffer as desired here.

C   Do not include packet header data—it is filled by this tool.

C   Fill first 2000 bytes with first packet data,

C       next 3000 bytes with second packet data,

C       last 4000 bytes with third packet data

C   Create simulated file

                returnstatus = pgs_io_l0_file_sim(

                                        spacecrafttag,
                                        appid,
                                        firstpacketnum,
                                        startutc,
                                        numvalues,
                                        timeinterval,
                                        datalength,
                                        filename,
                                        otherflags
                                        appdata,
                                        qualmisslen,
                                        qualdata,
                                        missdata)
```

NOTES: This tool is intended for use in science software development and testing, but not for production purposes.

When used to create file for EOS AM or EOS PM or EOS AURA (EDOS format) the Construction Record creation tool (PGS_IO_L0_EDOS_hdr_Sim()) must also be called to create the PDS/EDS Construction Record.

RELEASE NOTES:

This function conforms to EDOS-EGS ICD (June 28, 1996)

REQUIREMENTS: There is no SDP Toolkit requirement for this functionality. This tool was created to support internal ECS SDP Toolkit development and testing, and it is being provided as a service to the user.

6.2.1.2 HDF File I/O Tools

The ECS standard file format for transmission of datasets is The HDF Group's (THG's) Hierarchical Data Format (HDF). ECS has built extensions to HDF4 and HDF5, known as HDF-EOS and HDF-EOS5, which will support most recognized EOS era earth sciences data structures. Presently these data structures are grid, point and swath structures. If, in some cases, these are not sufficient, HDF could be used along with ECS metadata to specify an output file. Version 2.19 of HDF-EOS and version 1.15 of HDF-EOS5 are delivered with SCF Toolkit 5.2.19.

HDF-EOS (HDF-EOS5) is built on HDF4 (HDF5) low level functions and The HDF Group conventions were adhered to. The most prominent example is the user input of physical file handles. HDF requires physical handles, while the SDP toolkit requires logical handles. In order to make the toolkit compatible with HDF, the user will make one additional call to a process control function, obtain a physical handle and then open an HDF (HDF-EOS) file. Toolkit error handling functions may be used as necessary or desired. See the example in this section.

Important: HDF was designed to be a transport file format only, and support for such endeavors as updating a pre-existing file is very weak. Because of this and other performance considerations, HDF may not be the best choice of file format to use in internal processing of your files. We therefore strongly recommend that you use the Generic (Section 6.2.1.3) and Temporary (Section 6.2.1.6) I/O functions for internal processing, and reserve the use of HDF for initial read and final write of data products meant for archival and distribution.

EXAMPLE OF USAGE OF HDF FUNCTIONS

The following code fragments are simple examples of how the science software might use the SDP Toolkit logical-to-physical filename translation function in conjunction with the HDF4 open function. See Sections 6.2.2, 6.2.3, Appendices C and B.

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

```
? PRODUCT OUTPUT FILES
399|test10.hdf|/fire2/toma/data||||3
399|test9.hdf|/fire2/toma/data||||2
399|test8.hdf|/fire2/toma/data||||1
```

C

```
#include <PGS_PC.h>
#include <hdf.h>
#include <dfi.h>
#define HDF_INFILE 399
PGSt_integer version;
char physical_filename[PGSd_PC_FILE_PATH_MAX];
PGSt_SMF_status returnStatus;
int32 hdf_status;
int16 n_dds;
```

```
/*
Begin example
*/
version = 1;
returnStatus = PGS_PC_GetReference
      ( HDF_FILE, &version, physical_filename );
/*
Variable physical_filename now contains the string
"/fire2/toma/data/test10.hdf"
Variable version now contains the value 2, i.e., the number
of versions left in order, below this version in the PC file
*/
/*
Open the HDF file
*/
n_dds = 5; /* No. HDF data descriptor blocks */
hdf_status = Hopen(physical_filename,DFACC_CREATE,n_dds);
```

FORTRAN:

```
implicit none

INCLUDE           'PGS_SMF.f'
INCLUDE           'PGS_PC.f'
INCLUDE           'PGS_PC_9.f'
INTEGER           HDF_INFILE
PARAMETER         (HDF_INFILE=399)
CHARACTER*(*)     physicalfilename
INTEGER           pgs_pc_getreference
INTEGER           version
INTEGER           returnstatus
INTEGER           hdfstatus
INTEGER           ndds

C

C Begin example

C

  version = 1
  returnstatus = pgs_pc_getreference
        .           ( HDF_INFILE, version, physicalfilename )

C

C Variable physicalfilename now contains the string

C "/fire2/toma/data/test10.hdf"

C Variable version now contains the value 2, i.e., the number
```

```
C of versions left in order below this version in the PC file
C
C Open the HDF file
C
   ndds = 5    ! No. HDF data descriptor blocks
   hdfstatus = hopen(physicalfilename,DFACC_CREATE,ndds)
```

NOTES:

a. In order for this tool to work properly in the SCF environment, a Process Control File (PCF) must first be created by the science software developer. This file is part of the mechanism that maps the logical file identifiers in the science code to physical filenames. (This mapping will be performed by the scheduling subsystem in the DAAC environment.) See Section 4.2.2, "File Management," for further discussion. UNIX environment variable $PGS_PC_INFO_FILE must point to this file.

   In general, the PCF created by the user must follow the format given in Appendix C.

b. Currently, the Toolkit installation script installs HDF-4.2.10 and hdf5-1.8.12.

c. Functions that write error messages to a log file are now available. See the Status Message (SMF) tool section.

6.2.1.3 Generic File I/O Tools

This section includes tools for performing I/O functions on files that are not in the ECS standard format, i.e., HDF. The file open tools (Gen_Open and Gen_OpenF) are used by the science software to open miscellaneous files, which means any files that are not HDF, Level 0, ancillary, temporary or intermediate files (see sections 6.2.1.2, 6.2.1.1, 6.3.1, and 6.2.1.6). The file close tools (Gen_Close and Gen_CloseF) are used in science software to close these miscellaneous files, and also to close temporary and intermediate files.

The tools in this section are also used by other Toolkit functions, to access ancillary files (section 6.3.1), Level 0 files (section 6.2.1.1) and other miscellaneous files.

There are three items that apply to this entire subgroup of tools:

a. These tools only perform open and close functions on files. Reads, writes and other I/O functions are to be done by native C and FORTRAN I/O.

b. Due to file handle and other considerations it was not possible to bind FORTRAN to the C tools using the macro binding package. Unlike the rest of the Toolkit, these functions have separate FORTRAN versions.

c. Science software should use the PGS_IO_Temp_Open tool to open a temporary or intermediate file; see Section 6.2.1.6.

Special note regarding FORTRAN 90: Tools PGS_IO_Gen_OpenF and PGS_IO_Gen_Temp_OpenF now have FORTRAN 90 versions. These versions support two specific usages of the F90 OPEN function that are not supported in ANSI FORTRAN 77; they do not support all F90 options of OPEN. At Toolkit installation time, you select between F77 and F90, and the appropriate source code file is compiled; the function names are the same in both versions of FORTRAN. Options and text that apply only to FORTRAN 90 are marked in this document as ***F90 SPECIFIC***.

Open a Generic File (C Version)

NAME: **PGS_IO_Gen_Open( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Open(
        PGSt_PC_Logical             file_logical,
        PGSt_IO_Gen_AccessType      file_access,
        PGSt_IO_Gen_FileHandle      **file_handle,
        PGSt_integer                file_version)
```

FORTRAN: (not applicable)

DESCRIPTION: Upon a successful call, this function will provide the argument PGSt_IO_Gen_FileHandle to support other "C" library stream manipulation routines.

INPUTS: file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-8. File Access Type

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_Append Update	"a+"	Open file for reading and writing, to the end of existing file, or creating a new file; whole file can be read, but writing only appended

file_version-specific version of the logical file. (NOTE: this value will default to '1' for the interim delivery. Multiple file versions will be a capability in Toolkit 3 and later.)

OUTPUTS: file_handle-used to manipulate files with other "C" library stream I/O routines

RETURNS:

Table 6-9. PGS_IO_Gen_Open Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX system error
PGSIO_E_GEN_OPENMODE	Invalid access mode
PGSIO_E_GEN_FILE_NOEXIST	No entry for file logical ID in $PGS_PC_INFO_FILE
PGSIO_E_GEN_REFERENCE_FAILURE	Can not find physical file name with logical ID in $PGS_PC_INFO_FILE
PGSIO_E_GEN_BAD_ENVIRONMENT	Environment error reported by Process Control

(NOTE: the above are short descriptions only; full text of messages appears in files $PGSMSG/PGS_IO_1.t. Descriptions may change in future releases depending on external ECS design.)

EXAMPLE:

```
// This example illustrates how to open a Product Output
   File for writing //

PGSt_SMF_status         returnStatus;
PGSt_PC_Logical         logical;
PGSt_IO_Gen_AccessType  access;
PGSt_IO_Gen_FileHandle  *handle;
PGSt_integer            version;

logical = 10;
version = 1;            // will default to 1 for Toolkit 3 on out //
access = PGSd_IO_Gen_Write;
returnStatus = PGS_IO_Gen_Open( logical,access,&handle,
                                version );
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES: A file opened for write that already exists will be overwritten.

This function will support all POSIX modes of fopen.

While all modes of access are supported for this tool, those modes that allow for writing to a file (i.e., not PGSd_IO_Gen_Read) are intended for Toolkit access only. The only files that the science software should write to are product output files (HDF) and Temporary, or Intermediate files.

The only exceptions to this are for Support Output files that may need to be archived, but which are not considered to be products.

!!!!!!!!!! ALERT !!!!!!!!!!

During testing of this tool, the mode AppendUpdate (a+)!! 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.

IMPORTANT TOOLKIT 5 NOTES

The following environment variable MUST be set to assure proper operation:

PGS_PC_INFO_FILE path to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit as such:

PGS_PRODUCT_INPUT	path to standard input files
PGS_PRODUCT_OUTPUT	path to standard output files
PGS_SUPPORT_INPUT	path to supporting input files
PGS_SUPPORT_OUTPUT	path to supporting output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the '!' symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must be appear prior to any dependent subject entry.

The status condition PGSIO_E_GEN_BAD_ENVIRONMENT now indicates an error status on the global path statement as defined in the PCF, and NOT on an environment variable. However, as with previous releases, the status message associated with this condition may reference the above "tokens," but this is only to indicate which of the global path statements is problematic.

REQUIREMENTS: PGSTK-0360, PGSTK-1360

Open a Generic File (FORTRAN Version)

NAME: **PGS_IO_Gen_OpenF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:

```
INCLUDE    'PGS_SMF.f'
INCLUDE    'PGS_PC.f'
INCLUDE    'PGS_PC_9.f'
INCLUDE    'PGS_IO.f'
INCLUDE    'PGS_IO_1.f'

integer function pgs_io_gen_openf(file_logical, file_access,
                                  record_length, file_handle,
                                  file_version)

integer file_logical
integer file_access
integer record_length
integer file_handle
integer file_version
```

DESCRIPTION: Upon a successful call, this function will allocate a logical unit number to support FORTRAN READ and WRITE statements. This is returned to the user via the parameter file_handle. The user provides the logical file identifier and file version number, which internally get mapped to the associated physical file.

INPUTS: file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-10. File Access Type (1 of 2)

PGS FORTRAN Access Mode	Rd/Wr/Update/ Append	FORTRAN 77/90 'access='	FORTRAN 77/90 '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

Table 6-10. File Access Type (2 of 2)

PGS FORTRAN Access Mode	Rd/Wr/Update/ Append	FORTRAN 77/90 'access='	FORTRAN 77/90 'form='
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
F90 SPECIFIC			
PGSd_IO_Gen_AseqFrm	Append	Sequential	Formatted
PGSd_IO_Gen_AseqUnf	Append	Sequential	Unformatted

record_length-record length must be greater than 0 for direct access

F90 SPECIFIC must be greater than or equal to 0 for sequential access; if 0, file is opened with default record length

file_version-version of file to open (minimum value = 1)

OUTPUTS: file_handle-used to manipulate files READ and WRITE

RETURNS:

Table 6-11. PGS_IO_Gen_OpenF Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_NO_FREE_LUN	All logical unit numbers are in use
PGSIO_E_GEN_OPENMODE	Illegal open mode was specified
PGSIO_E_GEN_OPEN_OLD	Attempt to open with STATUS=OLD failed
PGSIO_E_GEN_OPEN_NEW	Attempt to open with STATUS=NEW failed
PGSIO_E_GEN_OPEN_RECL	Invalid record length specified
PGSIO_E_GEN_FILE_NOEXIST	File not found, cannot create
PGSIO_E_GEN_REFERENCE_FAILURE	Can't do Temporary file reference

EXAMPLE:

```
integer     returnstatus
integer     file_logical
integer     file_access
integer     record_length
integer     file_handle
integer     file_version

file_version      = 3
file_logical      = 101
file_access       = PGSd_IO_Gen_WSeqFrm
```

```
returnstatus = PGS_IO_Gen_OpenF( file_logical, file_access,
                                 record_length, file_handle,
                                 file_version)

if (returnstatus .NE. PGS_S_SUCCESS) then

C    goto 1000
end if
 .
 .
 .

1000 <error handling goes here>
```

NOTES:

While all modes of access are supported for this tool, those modes that allow for writing to a file (i.e., not PGSd_IO_Gen_Read) are intended for Toolkit access only. The only files that the science software should write to are product output files (HDF) and Temporary, or Intermediate files.

In order to ascertain the number of versions currently associated with the logical identifier in question, make a call to PGS_PC_Get_NumberOfFiles() first (Toolkit 3 and later.)

Due to the nature of FORTRAN IO, it is possible to write a file opened for reading as well as read a file opened for writing. The matching of access mode to IO statement cannot be enforced by the tool. This is up to the user.

Once a file has been opened with this tool, it must be closed with a call to PGS_IO_Gen_CloseF before being re-opened. Failure to do this will result in undefined behavior.

IMPORTANT TOOLKIT 5 NOTES

The following environment variable MUST be set to assure proper operation:

PGS_PC_INFO_FILE	path to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit as such:

PGS_PRODUCT_INPUT	path to standard input files
PGS_PRODUCT_OUTPUT	path to standard output files
PGS_SUPPORT_INPUT	path to supporting input file
PGS_SUPPORT_OUTPUT	path to supporting output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a

global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the '!' symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must be appear prior to any dependent subject entry.

It is error condition to have an input file specified in the PCF that does not exist on disk. The behavior of the tool is undefined when attempting to open such a file for reading.

REQUIREMENTS: PGSTK-0360

Close a Generic File, Temporary or Intermediate File (C Version)

NAME: **PGS_IO_Gen_Close( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Close(
        PGSt_IO_Gen_FileHandle     *file_handle);
```

FORTRAN: (not applicable)

DESCRIPTION: This tool closes a stream opened by a call to the "C" version of the Generic I/O Open tools.

INPUTS: fileHandle-file handle returned by PGS_IO_Gen_Open or PGS_IO_Gen_Temp_Open.

OUTPUTS: NONE

RETURNS:

Table 6-12. PGS_IO_Gen_Close Returns

Return	Description
PGS_S_SUCCESS	Success
PGSIO_E_GEN_CLOSE	Error closing file

EXAMPLES:

```
PGSt_IO_Gen_FileHandle  *handle;
PGSt_SMF_status         returnStatus;

returnStatus = PGS_IO_Gen_Close( handle );
if (returnStatus != PGS_S_SUCCESS)
{
   goto EXCEPTION;
}
else
{
   .
   .
   .
}
```

```
EXCEPTION:
```

NOTES: Usage of this tool is optional, but failure to close a file could result in loss of data, destroyed files, or possible intermittent errors in your program.

As a consequence of calling this tool, any data left unwritten in the output buffer will be flushed to the output stream; likewise, any data left unread in the input buffer will be discarded.

!!!!!!!!!! ALERT !!!!!!!!!!

Never attempt to close a file that has not been initialized, or previously used in an open call. Failure to heed this warning will result in program abort on many platforms.

REQUIREMENTS: PGSTK-0360

Close a Generic File (FORTRAN Version)

NAME: **PGS_IO_Gen_CloseF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:
```
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(file_handle)
integer file_handle
```

DESCRIPTION: This tool closes a FORTRAN IO unit opened by call to PGS_IO_Gen_OpenF or PGS_IO_Gen_Temp_OpenF.

INPUTS: file_handle-file handle returned by PGS_IO_Gen_OpenF or PGS_IO_Gen_Temp_OpenF

OUTPUTS: NONE

RETURNS:

Table 6-13. PGS_IO_Gen_CloseF

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_GEN_CLOSE	Attempt to close file failed
PGSIO_E_GEN_ILLEGAL_LUN	file_handle LUN was out-of-bounds
PGSIO_W_GEN_UNUSED_LUN	file_handle LUN was not in use

EXAMPLES:
```
integer     handle
integer     returnstatus

returnstatus = PGS_IO_Gen_CloseF(handle)
if (returnstatus != PGS_S_SUCCESS) goto 1000
.
.
.

100 <error handling goes here>
```

NOTES: Failure to close a file could result in loss of data, destroyed files, or possible intermittent errors in your program.

This tool expects the input file_handle to point to a file that was successfully opened via a call to either the tool PGS_IO_Gen_OpenF or the tool PGS_IO_Gen_Temp_OpenF. If this is not the case, the result of calling the tool is undefined.

REQUIREMENTS: PGSTK-0360

6.2.1.4 Metadata Tools

This set of tools is designed to manage the metadata that are generated with each EOS product, i.e., the granule-level metadata. The tools also provide a mechanism for populating the inventory data base tables with the metadata for each granule. The purpose of these tools is:

- To ensure that the metadata produced conforms to ECS standards in content and format; and
- To provide access files from within the science algorithms to metadata contained in input files.

The overall context of metadata in ECS, and further details on the use of the metadata tools are provided in Appendix J of this document.

The metadata tools in the SDP toolkit library are called from within a PGE to read and write metadata. The metadata attributes that will be assigned values during processing are identified in the metadata configuration file (MCF). The MCF is read into memory and toolkit calls are used to populate values for the attributes. When the metadata population process is complete, metadata "blocks" are written to product output files as HDF data objects called global attributes (not to be confused with individual metadata elements which are also called attributes). All output metadata is in object description language (ODL).

Multiple MCFs may be opened and written to from within a single PGE. The five metadata tools that are used in conjunction with MCFs must be called in a specific sequence, once for each MCF. First, each MCF must be initialized with **PGS_MET_Init**, which also assigns values for "system" metadata. Values generated within the PGE are assigned to attributes in the MCF using **PGS_MET_SetAttr** and/or **PGS_MET_SetMultiAttr**. To return the value of any metadata attribute in the MCF that has received a value **PGS_MET_GetSetAttr** may be used. After all values have been assigned, **PGS_MET_Write** is used to write the metadata to the product or, alternatively for non-HDF products, to a separate ASCII metadata file. Finally, **PGS_MET_Remove** frees up memory used by the MCFs. If the HDF file is of type HDF4 user may still call HDF's **SDstart** to open HDF file to write metadata. However, if the HDF file is of type HDF5 user must call **PGS_MET_SDstart** to open the file (this function can also be used to open HDF file of type HDF4). The file opened by **PGS_MET_SDstart** needs to be closed by a call to **PGS_MET_SDend** after writing metadata to it.

Two additional toolkit routines are used to read metadata values from within the PGE. These may be called independently of any MCF. **PGS_MET_GetPCAttr** may be used to return the value of metadata from input files identified to the process control (PC) system. **PGS_MET_GetConfigData** may be used to return the value of runtime metadata from the Process Control File.

The FORTRAN versions of PGS_MET_SetAttr, PGS_MET_SetMultiAttr PGS_MET_GetConfigData, PGS_MET_GetSetAttr, and PGS_MET_GetPCAttr must include an underscore and an extra character at the end of the function name to indicate the data type being handled: _S for string values, _I for integer and unsigned int values, and _D for single or double

precision real values. For example, the function PGS_MET_SetAttr actually represents three different FORTRAN functions:

- PGS_MET_SetAttr_S to set the value of string and datetime attributes
- PGS_MET_SetAttr_I to set integer and unsigned int values; and
- PGS_MET_SetAttr_D to set real or double values

As discussed in greater detail in Appendix J, two separate metadata blocks are handled by the metadata tools. These are called inventory and archive. Inventory consists of "core" attributes, i.e. those that are part of the ECS Data Model, which will reside in the ECS inventory tables and will thus be available to query on in locating granules. Archive metadata refers to metadata that a data producer wants to be included with the data granule, but need not be searchable by the system and will therefore not be used to populate the inventory tables. Archive metadata can, however, be read from HDF input files using toolkit calls.

The inventory and archive blocks are referenced in the toolkit calls by an array, e.g. mdHandles(n), where n=1 (for C, n=2 for FORTRAN) indicates inventory metadata and n=2 (or n=3 for FORTRAN) indicates archive metadata. To write an ASCII version of the metadata for non-HDF files mdHandles(0) (or n=1 for FORTRAN) is used to indicate that all metadata block are to be written together. It is possible to define other blocks and write them to HDF product output files or to ASCII metadata output files, but these will not be handled by the system. For example, if the granule is subsetted using ECS routines, only the inventory and archive blocks will be copied into the resultant file.

Additional description and extensive examples of the usage of MET tools can be found in the *HDF-EOS Users Guide for the ECS Project, Vol. 1*, Section 7 and 8.

A description of each MET tool follows:

Initialize a Metadata Configuration File (MCF) into Memory

NAME: **PGS_MET_Init()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Init(
        PGSt_PC_Logical          fileId,
        PGSt_MET_all_handles     mdHandles)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include  "PGS_SMF.h"

integer function pgs_met_init(fileId, mdHandles)

integer          fileId
character*       PGS_MET_GROUP_NAME_L
                 mdHandles(PGS_MET_NUM_OF_GROUPS)
```

DESCRIPTION: Initializes MCF file containing metadata.

INPUTS:

Table 6-14. PGS_MET_Init Inputs

Name	Description	Units	Min	Max
fileId	MCF file id	None	variable	variable

OUTPUTS:

Table 6-15. PGS_MET_Init Outputs

Name	Description	Units	Min	Max
mdHandles	metadata groups in MCF	None	N/A	N/A

RETURNS:

Table 6-16. PGS_MET_Init Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_LOAD_ERR	Unable to load <MCF> information. Lower level routines contain more information
PGSMET_E_GRP_ERR	Master groups are not supposed to be enclosed under any other group or object. The offending group is <name>
PGSMET_E_GRP_NAME_ERR	Group name length should not exceed PGS_MET_GROUP_NAME_L – 5.
PGSMET_E_NO_INVENT_DATA	Inventory data section not defined in the MCF
PGSMET_E_DUPLICATE_ERR	There is a another object with the same name for object <name> Duplicate names are not allowed within master groups
PGSMET_E_NUM)FMCF_ERR	Unable to load. The number of MCFs allocated has been exceeded.
PGSMET_E_PCF_VALUE_ERR	Metadata objects to be set from values defined in PCF could not be set. See error returns form the lower level routines. Initialization takes place nevertheless.

EXAMPLES:

C:

```
#include "PGS_MET.h"
#define     INVENTORYMETADATA 1
#define  MODIS_FILE 10253 /* This value must also be defined in
the PCF
      10253|hdftestfile|/home/asiyyid/pgetest/fortran/|||hdf
      testfile|1  :   */

#define  ODL_IN_MEMORY 0
int main()
{
PGSt_MET_all_handles handles;
char * fileName = "/home/modis/hdftestfile"; /* the user should
change this accordingly */
int32   hdfRet, sdid;
extern AGGREGATE PGSg_MET_MasterNode;
PGSt_SMF_status  ret = PGS_S_SUCCESS;
PGSt_integer    fileId = PGSd_MET_MCF_FILE;
PGSt_integer i;
double   dval, dval[6];
char* sval;
sval = (char*) malloc(30);
ret= PGS_MET_Init(fileId, handles);
if(ret != PGS_S_SUCCESS)
```

```
            {
          printf("initialization failed\n");
          return 0;
            }

          PGS_MET_Remove();
          printf("SUCCESS\n");
          return 0;
          }
```

FORTRAN:

```
          include "PGS_SMF.f"
          include "PGS_MET_13.f"
          include "PGS_MET.f"
C     the file id must also be defined in the PCF as follows
C     10253|hdftestfile|/home/asiyyid/pgetest/fortran/|||hd
C     testfile|1
          integer pgs_met_init
          integer MODIS_FILE
          parameter(MODIS_FILE = 10253)
          integer INVENTORYMETADATA
          parameter(INVENTORYMETADATA = 2)
          integer ODL_IN_MEMMORY
          parameter(ODL_IN_MEMMORY = 1)
C     the groups have to be defined as 49 characters long.
C     The C interface is 50.
C           The cfortran.h mallocs an extra 1 byte for the null
C           character '\0/', therefore making the actual length of a
C           string pass as 50.
          character*PGS_MET_GROUP_NAME_L
     1    mdHandles(PGS_MET_NUM_OF_GROUPS)
          character*50 fileName
          integer   result
          integer   pgs_met_init
          integer   hdfReturn
          double precision dval(1), dval(6)
          char*80   sval(5)
C           you must change this file spec in the PCF and the example
C           before running this example.
          fileName = "/home/asiyyid/pgetest/fortran/hdftestfile"
          result = pgs_met_init(PGSd_MET_MCF_FILE, groups)
          if(result.NE.PGS_S_SUCCESS) then
          print *, "Initialization error. See Logstatus for details"
          endif
```

```
print *, "SUCCESS"
end
```

NOTES: The MCF file must be in the format described in Appendix J.

Effective with the November 1996 SCF Toolkit release, multiple MCFs can now be initialized by repeated calls to this function.

REQUIREMENTS: PGSTK-0290, PGSTK-0370

Assign Values to Metadata Attributes

NAME: **PGS_MET_SetAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status

PGS_MET_SetAttr(
        PGSt_MET_handle    mdHandle,
        char               *attrNameStr,
        void               *attrValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
        include "PGS_MET.f"
        include "PGS_SMF.h"

integer function pgs_met_setattr(mdHandle, attrNameStr, attrValue)

character*(*)       mdHandle
character*(*)       attrName
'user defined'      attrValue
```

DESCRIPTION: After an MCF file is initialized into memory the user may assign values to metadata attributes using PGS_MET_SetAttr(). The values can be of following types and their array counterparts

PGSt_integer, PGSt_double, PGSt_real, char * (string)

INPUTS:

Table 6-17. PGS_MET_SetAttr Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group in MCF	none	N/A	N/A
attrNameStr	name.class of parameter	none	N/A	N/A
attrValue	value of attribute to be inserted	none	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-18. PGS_MET_SetAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_NESTED_OBJECTS	Object descriptions enclosing related objects must not be enclosed themselves by other objects
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to allocate memory
PGSMET_E_PARENT_GROUP	Multiple objects must have enclosing groups around them
PGSMET_E_CLASS_PARAMETER	Container object must also have class parameter defined
PGSMET_E_METADATA_CHILD	Metadata Objects are not allowed to enclose other objects
PGSMET_W_NOT_MULTIPLE	Object is not supposed to be multiple therefore resetting the value. The user may have given a class with the metadata name
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.
PGSMET_E_ILLEGAL_TYPE	Illegal type definition for metadata <attrName>. It should be a string
PGSMET_E_NO_DEFINITION	Unable to obtain <attr> of metadata <parameter> Either type or numval not defined
PGSMET_E_ILLEGAL_NUMVAL	Illegal NUMVAL definition for metadata <attrName>. It should be an integer
PGSMET_E_DD_UNKNOWN_PARM	The requested parameter <parameter name> could not be found in <agg node>
PGSMET_E_NEW_ODL_DATA_ERR	Unable to create a new odl <parameter>, probably due to lack of memory
PGSMET_E_INV_DATATYPE	Invalid data type definition in MCF for parameter <name>
PGSMET_E_INVALID_LOCATION	Invalid location for setting attribute value

EXAMPLES:

C:

```
/* For setting Inventory Attributes in the MCF */

/* NUMVAL i the MCF = 6 */

            dvals[0] = 10.0;
            dvals[1] = 20.0;
            dvals[2] = 30.0;
            dvals[3] = 40.0;
            dvals[4] = 50.0;
            dvals[5] = 60.0;
            ret = PGS_MET_SetAttr(handles[INVENTORYMETADATA],
                  "GRingPointLatitude.1", dvals);
```

```
 /* For setting Product Specific Attributes */

strcpy(informationname,"TestingAttribute1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"AdditionalAttributeName.1",&informationname);
strcpy(informationname,"testingAttributeValue1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
```

FORTRAN:

```
C  For setting Inventory Attributes in an HDF file

            dvals(1) = 10.0
            dvals(2) = 20.0
            dvals(3) = 30.0
            dvals(4) = 40.0
            dvals(5) = 50.0
            dvals(6) = 60.0
            ret =
            pgs_met_setattr_d(groups(INVENTORYMETADATA),
     1      “GRingPointLatitude.1", dvals)

C  For setting Product Specific Attributes

            informationname = "TestingAttribute1"
            ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
     1      "AdditionalAttributeName.1",informationname)
            informationname = "testingAttributeValue1"
            ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
     1      "ParameterValue.1",informationname)
```

NOTES: **1. Multiplicity:**

In TK5, a CLASS statement was introduced so that metadata objects with the same name could be distinguished from each other in the ODL tree. In TK5.1 this functionality was further extended to allow a single metadata object in the MCF to have multiple instances. This means that all the metadata objects within a master group in the MCF must have unique names. The use of the CLASS field in the name of a metadata attribute is optional and is needed only when the attribute in the MCF is within a group having a CLASS statement. See Appendix J for details and examples.

2. Nested Metadata:

There are certain metadata objects which are always described as a group of related metadata. To allow such groups to stay together in the MCF and the ODL tree, nested metadata objects are defined in the MCF using "Container Objects." in the MCF with related metadata as its child members. The child members are set individually as before. The container object does not have a value since it defines a concept and not an entity.

In the case of multiple container objects (e.g. there could be more than one instances of gring polygons), when a call to set a value of one of the child

metadata objects is made, it is the container object which is duplicated with a different class creating instances of all the child members. It is the users responsibility to set their values as well with subsequent call. Examples are given in Appendix J.

3. Array Filling:

TK5 imposed a restriction that metadata objects with values defined as arrays must be set with all the elements filled. This restriction is now lifted and the user has the freedom to set 1 to n values for a particular parameter where n is defined in the NUM_VAL field in the MCF. In this case where the values are being retrieved, the end of array is marked by:

INT_MAX	for integers
UINT_MAX	for unsigned integers
DBL_MAX	for doubles
NULL	char * (strings)

These values are defined in the limits.h and floats.h. Its analogous to null terminated strings defined as char[] arrays.

FORTRAN Users:

Use PGSd_MET_INT_MAX, PGSd_MET_DBL_MAX and PGSd_MET_STR_END respectively.

The user can check for these values to determine the actual number of values retrieved. In case where the number of values retrieved is equal to n, there is no end of array marker since user is expected to know n for setting the return buffer.

4. Permissible Data Locations:

PGS_MET_SetAttr can be used to assign values to metadata attributes which have DATA_LOCATION = “PGE”, “PCF”, or “TK”. Any attribute with DATA_LOCATION = “DSS”, “DAAC,” or “DP” can not be set by the PGE. An attempt to do so with PGS_MET_SetAttr will result in an error message of PGSMET_E_INVALID_LOCATION being generated in the runtime LOG file.

5. Metadata Types:

The tool provides a void interface through which different types of metadata can be set. The types supported are:

PGSt_integer
PGSt_uinteger
PGSt_double
string

and their arrays counterparts. PGSt_real has been omitted because of the changes in TK5.1.

It is very important that variable string pointers are used for string manipulations. This is because void interface is used. For example, the following piece of code would give an error or unexpected results:

```
.
.
char a[100];
.
.
strcpy(a, "MODIS");
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", a);
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a);
```

The first call is wrong because the routine expects char** but cannot force it because of void interface. The second call is wrong too because of the declaration of 'a' which is a constant pointer, i.e. it would always point to the same location in memory of 100 bytes. Only the following construct will work with the routine in which the string pointer is declared as a variable

```
char *a = "MODIS"
.
.
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a);
```

The above discussion is also true for arrays of strings. For example, the following is not allowed for the same reasons as above

```
.
.
char a[10][100];
.
.
strcpy(a[0], "MODIS");
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],
"SATELLITE_NAME", &a[0]);
```

while the following is acceptable:

```
.
.
char *a[10];
.
.
a[0] = "MODIS";
```

retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA], "SATELLITE_NAME", &a[0]);

IMPORTANT

The void buffer should always be large enough for the returned values otherwise routine behavior is uncertain.

REQUIREMENTS: PGSTK-0290 PGSTK-0410 PGSTK-380

Assign Mulitple Values to Metadata Attributes

NAME: **PGS_MET_SetMultiAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status

PGS_MET_SetMultiAttr(
        PGSt_MET_handle    mdHandle,
        char               *attrNameStr,

        PGSt_integer       num_val,
        void               *attrValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
        include "PGS_MET.f"
        include "PGS_SMF.h"

integer function pgs_met_setmultiattr(mdHandle, attrNameStr, numofval,
attrValue)

character*(*)      mdHandle
character*(*)      attrName
'user defined'     attrValue

integer            num_val
```

DESCRIPTION: After an MCF file is initialized into memory the user may assign multiple values to metadata attributes whose NUM_VAL is 1 in the MCF file using PGS_MET_SetMultiAttr(). This function sets the multi-value attribute and modifies NUM_VAL value to num_val passed to the function. The attribute values can be of following types and their array counterparts

PGSt_integer, PGSt_double, PGSt_real, char * (string)

INPUTS:

Table 6-19. PGS_MET_SetMultiAttr Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group in MCF	None	N/A	N/A
attrNameStr	name.class of parameter	None	N/A	N/A
num_val	number of values to be set by the user if NUM_VAL is 1 in the MCF	None	1	N/A
attrValue	value of attribute to be inserted	None	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-20. PGS_MET_SetMultiAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_NESTED_OBJECTS	Object descriptions enclosing related objects must not be enclosed themselves by other objects
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to allocate memory
PGSMET_E_PARENT_GROUP	Multiple objects must have enclosing groups around them
PGSMET_E_CLASS_PARAMETER	Container object must also have class parameter defined
PGSMET_E_METADATA_CHILD	Metadata Objects are not allowed to enclose other objects
PGSMET_W_NOT_MULTIPLE	Object is not supposed to be multiple therefore resetting the value. The user may have given a class with the metadata name
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.
PGSMET_E_ILLEGAL_TYPE	Illegal type definition for metadata <attrName>. It should be a string
PGSMET_E_NO_DEFINITION	Unable to obtain <attr> of metadata <parameter> Either type or numval not defined
PGSMET_E_ILLEGAL_NUMVAL	Illegal NUMVAL definition for metadata <attrName>. It should be an integer
PGSMET_E_DD_UNKNOWN_PARM	The requested parameter <parameter name> could not be found in <agg node>
PGSMET_E_NEW_ODL_DATA_ERR	Unable to create a new odl <parameter>, probably due to lack of memory
PGSMET_E_INV_DATATYPE	Invalid data type definition in MCF for parameter <name>
PGSMET_E_INVALID_LOCATION	Invalid location for setting attribute value

EXAMPLES:

C:

```
char *svals[5];
        PGSt_MET_all_handles handles;
        PGSt_integer num_val;
        char AttrName[256];
        char AttrValString[256];
        char *cptr;

        strcpy ( AttrName, "AdditionalAttributeName.1" );
        strcpy ( AttrValString, "string 1" );
        cptr = AttrValString;
        ret = PGS_MET_SetAttr ( handles[INVENTORYMETADATA], AttrName, &cptr );

        strcpy ( AttrName, "ParameterValue.1" );
        svals[0] = (char *) malloc(30);
        svals[1] = (char *) malloc(30);
        svals[2] = (char *) malloc(30);
        svals[3] = (char *) malloc(30);
        svals[4] = NULL;
```

```
        strcpy(svals[0], "Astring 11");
        strcpy(svals[1], "Astring 22");
        strcpy(svals[2], "Astring 33");
        strcpy(svals[3], "Astring 44");
        num_val = 6;
      ret = PGS_MET_SetMultiAttr(handles[INVENTORYMETADATA], AttrName,
num_val, svals);
```

FORTRAN:

```
      IMPLICIT NONE

      INCLUDE 'PGS_SMF.f'
      INCLUDE 'PGS_MET.f'
      include 'PGS_MET_13.f'
      INCLUDE 'PGS_PC.f'

      INCLUDE 'hdf.inc'

      integer PGS_MET_Init
      integer PGS_MET_SetAttr_s
      integer PGS_MET_SetMultiAttr_s
      character*50 svals2(5)
      character*(PGSd_MET_GROUP_NAME_L)
     +     mdHandles(PGSd_MET_NUM_OF_GROUPS) ! metadata group in MCF
      character*256 AttrName
      character*256 AttrValString
      integer status
      integer num_val

      integer INVENTORY
      PARAMETER (INVENTORY = 2 )
      integer MCF_FILE
      PARAMETER (MCF_FILE = 10250 )

      status = PGS_MET_Init ( MCF_FILE, mdHandles )
      AttrName = "AdditionalAttributeName.1"
      AttrValString = "string 2"
      status = PGS_MET_SetAttr_s ( mdHandles(INVENTORY), AttrName,
     &     AttrValString)

      AttrName = "ParameterValue.1"
      svals2(1) = "Astring 11"
      svals2(2) = "Astring 22"
      svals2(3) = "Astring 33"
      svals2(4) = "Astring 44"
      svals2(5) = PGSd_MET_STR_END
      num_val = 6

      status = PGS_MET_SetMultiAttr_s( mdHandles(INVENTORY), AttrName,
     &     num_val, svals2)
```

NOTES: See notes for PGS_MET_SetAttr

REQUIREMENTS: PGSTK-0290 PGSTK-0410 PGSTK-380

Accesses Metadata Attributes Already Set in Memory

NAME: **PGS_MET_GetSetAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetSetAttr(
PGSt_MET_handle    mdHandle,
        char*          attrNameStr,
        void*          attrValue)
```

FORTRAN:

```
include        "PGS_MET_13.f"
include        "PGS_MET.f"
include        "PGS_SMF.h"

integer function pgs_met_getsetattr(mdHandle, attrNameStr, attrValue)
character*     mdHandle
character*     attrName
'user defined' attrValue
```

DESCRIPTION: The MCF is first initialized into memory and some of the parameters are automatically set and some are set by the user using PGS_MET_SetAttr(). This tool is used to retrieve these values.

INPUTS:

Table 6-21. PGS_MET_GetSetAttr Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group	none	N/A	N/A
attrName	name.class of parameter	none	N/A	N/A

OUTPUTS:

Table 6-22. PGS_MET_GetSetAttr Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-23. PGS_MET_GetSetAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_DD_UNKNOWN_PARM	The requested parameter <parameter name> could not Be found in <agg node>
PGSMET_W_METADATA_NOT_SET	The metadata <name> is not yet set
PGSMET_E_NO_DEFINITION	Unable to obtain <attr> of metadata <parameter>
	Either NUM_VAL or type is not defined
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.

EXAMPLES:

C:

```
/* For accessing Inventory Attributes in an HDF file */

            for(i = 0; i < 6; i++) dvals[i] = 0.0;
            ret = PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
            "GRingPointLatitude.1", dvals);
            for(i = 0; i < 6; i++) printf("%lf", dvals[i]);
            printf("\n");

/* For accessing Product Specific Attributes in an HDF file */
            strcpy(sval," ");
            ret=PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
            "AdditionalAttributeName.1",&sval);

            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");
            strcpy(sval," ");
            "ParameterValue.1",&sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");
FORTRAN:

C For accessing Inventory Attributes in an HDF file

            dvals(1) = 0.0
            dvals(2) = 0.0
            dvals(3) = 0.0
            dvals(4) = 0.0
            dvals(5) = 0.0
            dvals(6) = 0.0

            ret = pgs_met_setattr_d(groups[INVENTORYMETADATA],
     1      "GRingPointLatitude.1", dvals)
            print *, dvals(1), dvals(2), dvals(3), dvals(4),
     1       dvals(5), dvals(6)
```

```
C For accessing Product Specific Attributes in an HDF file

            sval = " "
            ret=pgs_met_setattr_s(groups[INVENTORYMETADATA],
     1      "AdditionalAttributeName.1",sval)
            print *, sval
            sval = " "
            ret=pgs_met_setattr_s(groups[INVENTORYMETADATA],
     1      "ParameterValue.1",sval)
            print *, sval
```

NOTES: (See notes 1,2,3, and 4 in PGS_MET_SetAttrib()

REQUIREMENTS: PGSTK-0290 PGSTK-380

Accesses Metadata Parameters in HDF Products or Independent ASCII Files

NAME: **PGS_MET_GetPCAttr()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetPCAttr(
PGSt_PC_Logical  fileId,
PGSt_integer     version,
        char *          hdfAttrName,
        char *          parmName,
        void *          parmValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_getpcattr(fileId, version, hdfAttrName, parmName,
parmValue)

character*          fileId
integer             version
character*          hdfAttrName
character*          parmName
'user defined'      parmValue
```

DESCRIPTION: Metadata parameters held in HDF attributes or in a separate ASCII file can be read using this tool

INPUTS:

Table 6-24. PGS_MET_GetPCAttr Inputs

Name	Description	Units	Min	Max
fileId	product file id	none	Variable	variable
version	product version number	none	1	variable
hdfAttrName	name of HDF attribute containing metadata	none	N/A	N/A
parmName	metadata parameter name	none	N/A	N/A

OUTPUTS:

Table 6-25. PGS_MET_GetPCAttr Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-26. PGS_MET_GetPCAttr Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_PCREAD_ERR	"Unable to obtain <filename or attribute filename> from the PC table" Most likely that <filename or attribute filename> is not defined in the PCF
PGSMET_E_FILETOODL_ERR	"Unable to convert <filename> into an ODL format" error returns from lower level routines should explain the problem
PGSMET_E_AGGREGATE_ERR	Unable to create ODL aggregate <aggregate name> It Definitely means that ODL routine has failed to allocate enough Memory
PGSMET_E_SYS_OPEN_ERR	Unable to open pc attribute file Usually if the file does not exist at the path given, check the name and path of the file
PGSMET_E_ODLTOVAL_ERR	Unable to convert attribute values from the ODL format error Returns from lower level routines should explain the problem
PGSMET_E_NULL_PARAMETER	The requested parameter is a null value
PGSMET_E_NOT_SET	The requested parameter is not set

EXAMPLES:

```
C:
      char grpName[100];

/* For accessing Inventory Attributes in an HDF file */

            for(i = 0; i < 6; i++) dvals[i] = 0.0;
            ret = PGS_MET_GetPCAttr(MODIS_FILE, 1, "coremetadata",
            "GRingPointLatitude.1", dvals);
            for(i = 0; i < 6; i++) printf("%lf", dvals[i]);
            printf("\n");

/* For accessing Product Specific Attributes in an HDF file */

            strcpy(sval," ");
            ret=PGS_MET_GetPCAttr(MODIS_FILE,1,"coremetadata",
            "TestingAttribute1",&sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

/* For accessing attributes in the ASCII Metadata file */
/* NOTE: For retrieving attribute values from the ASCII metadata file, users
have to generate a group name first before calling the function
PGS_MET_GetPCAttr. The procedures are as follows:
```

```
1:
      In this case the group name is INVENTORYMETADATA
      sprintf(grpName, "%s%s", PGSd_MET_GROUP_STR, "INVENTORYMETADATA");
2:
      ret = PGS_MET_GetPCAttr(10268, 1, grpName, "REPROCESSINGPLANNED",
      &sval);

*/

            strcpy(sval," ");
            sprintf(grpName, "%s%s", PGSd_MET_GROUP_STR,
      "INVENTORYMETADATA");
            ret = PGS_MET_GetPCAttr(10268, 1, grpName,
      "REPROCESSINGPLANNED",  &sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

/* For LandSat7 Metadata output file */
/* NOTE: For retrieving the attribute from the Landsat7  meta file, users have
to generate a group name first before calling the function PGS_MET_GetPCAttr.
The procedures are as follows:

1:
      In this case the group name is "FORMAT_SUBINTERVAL_METADATA_1"
      sprintf(grpName,"%s%s",PGSd_MET_LSAT_GRP_STR,
      "FORMAT_SUBINTERVAL_METADATA_1");

2:
      ret = PGS_MET_GetPCAttr(10269, 1, grpName,
      "CONTACT_PERIOD_START_TIME", &sval);

*/
            strcpy(sval," ");
            sprintf(grpName,"%s%s",PGSd_MET_LSAT_GRP_STR,
      "FORMAT_SUBINTERVAL_METADATA_1");
            ret = PGS_MET_GetPCAttr(10269, 1, grpName,
      "CONTACT_PERIOD_START_TIME", &sval);
            for(i = 0; i<1; i++) printf("%s", sval);
            printf("\n");

FORTRAN:

            char grpName[100];

C For accessing Inventory Attributes in HDF file

            for(i = 0; i < 6; i++) dvals(i) = 0.0
            ret = pgs_met_getpcattr_d(MODIS_FILE, 1, "coremetadata",
     1       "GRingPointLatitude.1", dvals)
            print *, dval(1), dval(2), dval(3), dval(4), dval(5),
     1       dval(6)

C For accessing Product Specific Attributes in HDF file

            sval = " "
            ret=pgs_met_getpcattr_s(MODIS_FILE, 1, "coremetadata",
     1       " TestingAttribute1",&sval)
            print *, sval
```

```
C For accessing attributes in ASCII Metadata file

            sval = " "
            ret = pgs_met_getpcattr_s(10268, 1, grpName,
     1       "REPROCESSINGPLANNED", &sval)
            print *, sval

C For Landsat7 Metadata file
            sval = " "
            grpName(1:)=PGSd_MET_LSAT_GRP_STR//
     1      "FORMAT_SUBINTERVAL_METADATA_1"
            ret = pgs_met_getpcattr_s(10269, 1, grpName,
     1       "CONTACT_PERIOD_START_TIME", &sval
            print *, sval
```

NOTES: See Notes 1,2,3, and 4 in PGS_MET_SetAttr

In the ECS production environment all input files are accompanied by an ASCII version of the metadata (the .met file) so PGS_MET_GetPCAttr will always read metadata from the .met file. In the SCF environment if the data input file is in HDF a .met file need not be present and the metadata can be read from the file itself. This is an example of how an HDF input file should be designated in the PCF:

```
10253|hdfinputfile|/my/product/directory/|||hdfinputfile|1
```

The file names in the second and sixth fields must be identifal. If the input file is not in HDF, the metadata will be read from an ASCII file which must be separately identified in the sixth field of the input product entry of the PCF, as shown in this example:

```
10253|inputfile|/my/product/directory/|||inputfile.met|1
```

The .met file must have the same name as the product input file, with the .met extension appended. This file must be placed in the same directory as the input file.

Effective with the November 1996 SCF Toolkit delivery, the separate ASCII file can now be in the same format as the output from PGS_MET_Write().

In the ECS production environment the ASCII metadata file that accompanies a data input file delivered by Science Data Server does not contain archive metadata. For this reason, archive metadata can only be read from input files that are in HDF. If used to read a value for a metadata attribute that is contained in an HDF global text attribute named “archivemetadata” or “productmetadata” PGS_MET_GetPCAttr will attempt to read the metadata from the HDF file, even though an ASCII .met file is present. In all other cases, PGS_MET_GetPCAttr reads the ASCII .met file.

The ASCII file may be in one of two formats; either that written out by the PGS_MET_Write() routine or simple parameter=value construct. These formats are shown below for a simple case

```
OBJECT = SOMEPARAMETER
        NUM_VAL = 1
        VALUE = 200
END_OBJECT = SOMEPARAMETER
```

or

```
SOMEPARAMETER = 200
```

Note that if a parameter appears twice in the ASCII file (with the same parameter name and Class extension) only the first occurrence will be returned.

REQUIREMENTS: PGSTK-0290 PGSTK-0235

Accesses Configuration Data in the PC Table

NAME: **PGS_MET_GetConfigData()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_GetConfigData(
        char*           attrName,
        void*           attrValue)
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_met_getconfigdata( attrName, attrValue)
character*      attrName
'user defined'  attrValue
```

DESCRIPTION: Certain configuration parameters are held in the PC table as follows

```
10220|REMOTEHOST|sandcrab
```

This tool would retrieve the value "sandcrab" from the PC table given the name of the parameter "REMOTEHOST". The parameter id 10220 is not used here. The value string (e.g.. sandcrab) is assumed to be in ODL format and therefore different types are supported.

INPUTS:

Table 6-27. PGS_MET_GetConfigData Inputs

Name	Description	Units	Min	Max
attrName	name of parameter in PCF	none	N/A	N/A

OUTPUTS:

Table 6-28. PGS_MET_GetConfigData Outputs

Name	Description	Units	Min	Max
attrValue	value of attribute to be passed back to the user	none	N/A	N/A

RETURNS:

Table 6-29. PGS_MET_GetConfigData Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_AGGREGATE_ERR	"Unable to create ODL aggregate <aggregate name>" This should never occur unless the process runs out of memory
PGSMET_E_CONFIG_VAL_STR_ERR	"Unable to obtain the value of configuration parameter <name> from the PCF file". Likelihood is that either the parameter does not exist in the PCF or the PCF itself is in error which can be tested using pccheck.
PGSMET_E_CONFIG_CONV_ERR	"Unable to convert the value of configuration parameter <name> from the PCF file into an ODL format". Its most likely that the string values is not in ODL format.

EXAMPLES:

C:

```
/* These values must be defined in the PCF otherwise error is returned
*/
            ret = PGS_MET_GetConfigData("REV_NUMBER", &ival);
            strcpy(datetime, "");
            ret = PGS_MET_GetConfigData("LONGNAME", &datetime);
            dval = 0;
            ret = PGS_MET_GetConfigData("CENTRELATITUDE", &dval);
            printf("%d %lf %s\n", ival, dval, datetime);
```

FORTRAN:

```
C Retrieve some values from the PCF files. These must be
C defined in the PCF, otherwise the routine would return error
C Note the way _i for integer, _d for double and _s for strings are used
C at the end of the function name. This is necessary because fortran
C compiler would complain about type conflicts if a generic name
C is used
            ret = pgs_met_getconfigdata_i("REV_NUMBER", ival)
            datetime = ""
            ret = pgs_met_getconfigdata_s("LONGNAME", datetime)
            dval = 0
            ret = pgs_met_getconfigdata_d("CENTRELATITUDE", dval)
            if(ret.NE.PGS_S_SUCCESS) then
            print *, "GetConfigData failed.
            endif
```

```
print *, ival, dval, datetime
```

NOTES: See Notes 1, 2, 3, and 4 for PGS_MET_SetAttr().

Although This tool ignores the first field in the PCF file depicting the config id, it is still important that this field is unique for the PC utility to function correctly User is responsible for the returned buffers to be large enough to hold the returned values.

Addendum for TK5.1

This routine now simply retrieves the values from the PCF and does not perform type and range checking. The user is still required to assign enough space for the returned values.

REQUIREMENTS: PGSTK-0290 PGSTK-0380

Write Metadata and their Values to HDF Attributes and/or ASCII Output Files

NAME: **PGS_MET_Write()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Write(
        PGSt_MET_handle    mdHandle,
        char *             hdfAttrName,
        PGSt_integer       hdfFileId)
```

FORTRAN:

```
include 'PGS_MET_13.f'
include 'PGS_MET.f'
include 'PGS_SMF.h'

integer function pgs_met_write(mdHandle, hdfAttrName, hdfFileId)

        character*   mdHandle
        character*   hdfAttrName
        integer      hdfFileId
```

DESCRIPTION: This is the final tool that PGE uses when all the metadata parameters are set in memory. The tool checks that all the mandatory parameters are set.

INPUTS:

Table 6-30. PGS_MET_Write Inputs

Name	Description	Units	Min	Max
mdHandle	metadata group in MCF	none	N/A	N/A
hdfAttrName	HDF attribute name to contain metadata	none	N/A	N/A
hdfFileId	HDF file ID	none	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-31. PGS_MET_WriteReturns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_NO_INITIALIZATION	Metadata file is not initialized
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to malloc memory space
PGSMET_E_GROUP_NOT_FOUND	No group called <name> found in the MCF
PGSMET_E_OPEN_ERR	Unable to open <temporary> file with file id <fileId>
PGSMET_E_SD_SETATTR	Unable to set the HDF file attribute. Note: HDF4.0r2 and Previous versions of HDF have imposed a limit.
PGSMET_E_MALLOC_ERR	Unable to allocate memory for the hdf attribute
PGSMET_E_MAND_NOT_SET	Some of the mandatory parameters were not set
PGSMET_E_FGDC_ERR	Note: HDF attribute is still written out. Unable to convert UTC Input date time string to FGDC values
PGSMET_E_ILLEGAL_HANDLE	Handle is illegal. Check that initialization has taken place.
PGSMET_E_HDFFILENAME_ERR	Unable to obtain HDF filename.
PGSMET_E_ASCII_ERR	Unable to open MET ASCII file.

EXAMPLES:

C:

```
/* Write to ASCII metadata file for non-HDF output product */
            ret= PGS_MET_Write(handles[ODL_IN_MEMMORY],NULL, 101);
            if(ret != PGS_S_SUCCESS)
                  {
                  printf("ASCII Write failed\n");
                  }
/* Write to HDF file */
            ret= PGS_MET_Write(handles[INVENTORYMETADATA],  "metadata", sdid);
            if(ret != PGS_S_SUCCESS)
                   {
                   printf("HDFWrite failed\n");
                   }
```

FORTRAN:

```
C Write to ASCII file for non-HDF output product
                  result= pgs_met_write(groups(ODL_IN_MEMORY),dummyStr, 101)
                  if(result.NE.PGS_S_SUCCESS.AND.
                  result.NE.PGSMET_MAND_NOT_SET) then
     1            print *,"ASCII Write failed"
                  endif
C Write to HDF file
                  result= pgs_met_write(groups(INVENTORYMETADATA),
```

```
1            "coremetadata", sdid)
             if(result.NE.PGS_S_SUCCESS.AND. result.NE.PGSMET_MAND_NOT_SET)
             then
1            print *,"ASCII Write failed"
             endif
```

NOTES: When writing an attribute which has been defined as "UNSIGNED INT", the value written to the ASCII or HDF file may appear negative. The user should use the type “unsigned int” or the ECS equivalent (PGSd_uinteger) to interpret the value correctly. (see Note 4 of PGS_MET_SetAttr in Section 6.2.1.4.)

This routine can be used multiple times to write/attach separate master groups as local or global HDF attributes. To attach a mastergroup to a local element in an HDF file, an sds_id must be passed in as an argument, rather than an sd_id(hdfFileId). **!!!NOTE!!!** : Attaching metadata to a local element using the Toolkit is not standard practice for HDF-EOS files and should be avoided.

When writing the inventory metadata (MASTERGROUP = INVENTORYMETADATA in the MCF, mdHandle = coremetadata in the function call) to an HDF file, an ASCII version of the metadata is automatically created in the data product output directory. It is given the same name as the data product output, with the extension .met, i.e. ProductName.met. If the data product output is not in HDF, the following lines must be included in the PCF in order to create this required .met file:

```
?PRODUCT OUTPUT

100|ProductName|my/output/directory|||1
.
.
.
? USER RUNTIME PARAMETERS

101|ProductMetadataFile|100:1
```

where the second field is simply a comment.

An ASCII version of the metadata file will be created in the execution directory with the name *ProductName.met*. The user needs to call PGS_MET_Write with mdHandle[0], the HDF attribute name set to NULL and the identifier set to the logical identifier in the PCF.

2. If MANDATORY parameters are not set, an error PGSMET_E_MAND_NOT_SET is returned only in a PGE. The value of the metadata is set to as follows:

DATA_LOCATION	VALUE
PGE	"NOT SET"
PCF	"NOT FOUND"
MCF	"NOT SUPPLIED"

TK	“NOT OBTAINED”
DSS	“NOT PROVIDED”
DAAC	“NOTSUPPORTED”
DP	“NOT INCLUDED”

The writing of the hdf header is not affected

NOTE: A warning PGSMET_W_METADATA_NOT_SET is issued if MANDATORY has the value FALSE in the MCF, and the specific attribute will not appear in the HDF-EOS attribute or the ASCII file.

3. Only system errors such as memory failure, file openings etc. should be able to abort the write procedure.

4. NUM_VAL and CLASS fields are written in the HDF header

For metadata of type DATETIME, additional metadata is produced:

CALENDATDATETIME becomes CALENDARDATE and TIMEOFDAY.

RANGEBEGININGDATETIME becomes RANGEBEGININGDATE and RANGEBEGININGTIME

RANGEENDINGDATETIME becomes RANGEENDINGDATE and RANGEENDINGTIME

The user no longer has to worry about the size of the MCF exceeding the HDF limit on attribute sizes. This is now handled internally. The user simply needs to set coremetadata (or archivemetadata) and if the limit is exceeded, coremetadata.0, .1, etc. are produced.

5. With the release 5.2.16 of TOOLKIT users can get core metadata in XML format in addition to the ODL format. To get both *.met and *.xml files user need to modify their PCF file adding 3 lines

```
10260|XMLstylesheet.temp|||||1

10303|science.xsl|~/database/common/MET||||1

10256|XML METADATA GENERATION FLAG; 0=no, 1=yes|0
```

(as shown in the template PCF file in the runtime directory of TOOLKIT) and setting XML

flag to 1. If PCF does not include the line

10256|XML METADATA GENERATION FLAG; 0=no, 1=yes|0

or flage is set to zero (as above), toolkit should work as in previous versions, creating only ODL metadata. When XML flag is set to 1 in the PCF file, TOOLKIT will produce *.xml besides the *.met file for INVENTORY metadata

and also will write XML metadata into the HDF file in the "xmlmetadata" global attribute as for the coremetadata.

REQUIREMENTS: PGSTK-0290, PGSTK-0380, PGSTK-0400, PGSTK-0450, PGSTK-0510

Free Memory of MCFs

NAME: **PGS_MET_Remove()**

SYNOPSIS:

C:

```
#include "PGS_MET.h"

PGSt_SMF_status
PGS_MET_Remove()
```

FORTRAN:

```
include "PGS_MET_13.f"
include "PGS_MET.f"
include "PGS_SMF.h"

integer function pgs_met_remove()
```

DESCRIPTION: This routine removes ODL representation of all MCF files and some internal files used by the MET tools.

INPUTS: None

OUTPUTS: None

RETURNS: None

EXAMPLES:

C:

```
result = PGS_MET_Remove();
printf("SUCCESS\n");
return 0;
```

FORTRAN:

```
print *, ival, dval, datetime
result = pgs_met_remove()
print *, "SUCCESS"
end
```

NOTES: This routine must be called by the user before the program terminates.

REQUIREMENTS: None

Open HDF File of Type HDF4 or HDF5 for Writing Metadata

NAME: **PGS_MET_SDstart()**

SYNOPSIS:

C:

```
#include <PGS_MET.h>

PGSt_SMF_status
PGS_MET_SDstart(
char     *filename,
uintn    access_mode,
PGSt_integer  *HDFfid)
```

FORTRAN:

```
include  'PGS_SMF.f'
include 'PGS_MET.f'

integer function pgs_met_sfstart(filename, access_mode, hdffid)

character*(*) filename
integer hdffid
```

DESCRIPTION: This tool opens the HDF files of type HDF4 and/or HDF5 and initializes the SD inetface.

INPUTS:

Table 6-32. PGS_MET_SDstart Inputs

Name	Description	Units	Min	Max
filename	HDF file name (with full path)	none	variable	variable
access_mode	Access mode for opening HDF files. It can be: HDF5_ACC_RDONLY, HDF5_ACC_RDRW, HDF5_ACC_CREATE for HDF5files and HDF4_ACC_RDONLY, HDF4_ACC_RDWR, HDF4_ACC_CREATE for HDF4 files	none		

OUTPUTS:

Table 6-33. PGS_MET_SDstart Outputs

Name	Description	Units	Min	Max
HDFfid	SD id of the file opened	none	N/A	N/A

RETURNS:

Table 6-34. PGS_MET_SDstart Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_HDF5_FILE_TYPE_ERROR	Cannot determine whether the file is hdf4, hdf5, or none-hdf type
PGSMET_E_SD_START	File <filename> is not HDF type and cannot be opened
PGSMET_E_SD_START	Cannot open HDF5 file <filename>
PGSMET_E_SD_START	Cannot open HDF4 file <filename>

EXAMPLES:

C:

```
PGSt_SMF_status   retstatus;
PGSt_integer      Sdid;
retstatus = PGS_MET_SDstart( "/home/username/myhdf.h5",
                                HDF5_ACC_RDWR, &SDid);
if (retstatus != 0)
{
*** do some error handling ***
      :
      :
}
```

FORTRAN:

```
      implicit none
      integer sdid
      integer status
      status = PGS_MET_SFstart("/home/username/myhdf.h5",
     *                         HDF5_ACC_RDWR, sdid)
      if(status .ne. 0) goto 999
```

NOTES: None

Close HDF file of Type HDF4 or HDF5

NAME: **PGS_MET_SDend()**

SYNOPSIS:

C:

```
#include <PGS_MET.h>

PGSt_SMF_status
PGS_MET_SDend(cha PGSt_integer  HDFfid)
```

FORTRAN:

```
include  'PGS_SMF.f'
include 'PGS_MET.f'

integer function pgs_met_sfend(hdffid)
integer hdffid
```

DESCRIPTION: This tool closes the HDF files of type HDF4 and/or HDF5 that have been opened by calling PGS_MET_SDstart.

INPUTS:

Table 6-35. PGS_MET_SDend Outputs

Name	Description	Units	Min	Max
HDFfid	SD id of the file opened	none	N/A	N/A

OUTPUTS: None

RETURNS:

Table 6-36. PGS_MET_SDend Returns

Return	Description
PGS_S_SUCCESS	
PGSMET_E_SD_END	Cannot close the HDF file with ID <sd id>

EXAMPLES:

C:

```
PGSt_SMF_status   retstatus;
PGSt_integer      Sdid;
retstatus = PGS_MET_SDend( SDid);

if (retstatus != 0)

{

*** do some error handling ***

      :

      :

}
```

FORTRAN:

```
implicit none
integer sdid
integer status
status = PGS_MET_SFend( sdid)

if(status .ne. 0) goto 999
```

NOTES: None

6.2.1.5 Data Quality Assurance

The tools in this section will be used to support the analysis of Q/A data output from the production processes. There is no Toolkit tool to meet this requirement, however, this requirement is being met by other HDF functionality

REQUIREMENTS: PGSTK-0510

6.2.1.6 Temporary and Intermediate Files

This section contains descriptions of tools that are specific to temporary and intermediate file I/O. A temporary file is a file that exists only for the duration of a single PGE; it is deleted following successful PGE termination. An intermediate file exists for a user-defined time after the PGE terminates.

After you open a temporary or intermediate file, use the native C or FORTRAN I/O routines to perform I/O.

Note that there are no “Temp_Close” tools; use the Gen_Close tools to close files. See “Generic File I/O Tools” (Section 6.2.1.3).

Special note regarding FORTRAN 90: Tools PGS_IO_Gen_OpenF and PGS_IO_Gen_Temp_OpenF now have FORTRAN 90 versions. These versions support two specific usages of the F90 OPEN function that are not supported in ANSI FORTRAN 77; they do not support all F90 options of OPEN. At Toolkit installation time, you select between F77 and F90, and the appropriate source code file is compiled; the function names are the same in both versions of FORTRAN. Options and text that apply only to FORTRAN 90 are marked in this document as ***F90 SPECIFIC***.

IMPORTANT CHANGES FROM TOOLKIT 4

The following environment variables MUST be set to assure proper operation:

PGS_PC_INFO_FILE	path to process control file

However, the following environment variables are NO LONGER recognized by the Toolkit:

PGS_TEMPORARY_IO	path to temporary files
PGS_INTERMEDIATE_INPUT	path to intermediate input files
PGS_INTERMEDIATE_OUTPUT	path to intermediate output files

Instead, the default paths, which were defined by these environment variables in previous Toolkit releases, may now be specified as part of the Process Control File (PCF). Essentially, each has been replaced by a global path statement for each of the respective subject fields within the PCF. To define a global path statement, simply create a record that begins with the ‘!’ symbol defined in the first column, followed by the global path to be applied to each of the records within that subject field. Only one such statement can be defined per subject field and it must appear prior to any dependent subject entry.

The status condition PGSIO_E_GEN_BAD_ENVIRONMENT now indicates an error status on the global path statement as defined in the PCF, and NOT on an environment variable. However, as with previous releases, the status message associated with this condition may reference the above “tokens,” but this is only to indicate which of the global path statements is problematic.

“The environment variable PGS_HOST_PATH, formerly used to direct the Toolkit to the location of the internet protocol address for the local host, has been replaced by PDPS functionality which can perform this function in more effective manner. For this reason, the use of this environment variable is no longer supported. **FAILURE TO HEED THIS WARNING MAY RESULT IN UNPREDICTABLE RESULTS FOR THE PGE**. To properly emulate the manner in which the PDPS system provides this information to the Toolkit, continue to use the runtime parameter PGSd_IO_Gen_HostAddress to advertise the IP address of the local host.”

Open a Temporary/Intermediate File (C Version)

NAME: **PGS_IO_Gen_Temp_Open( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Temp_Open(
        PGSt_IO_Gen_Duration      file_duration,
        PGSt_PC_Logical           file_logical,
        PGSt_IO_Gen_AccessType    file_access,
        PGSt_IO_Gen_FileHandle**  file_handle);
```

FORTRAN: (not applicable)

DESCRIPTION: This routine lets the user create and open Temporary and Intermediate files with a variety of access modes. The returned argument PGSt_IO_Gen_FileHandle is directly compatible with the standard "C" library stream I/O manipulation routines.

INPUTS: file_duration:

```
PGSd_IO_Gen_Endurance       // Creates Intermediate File //
PGSd_IO_Gen_NoEndurance     // Creates Temporary File  //
```

file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-37. File Access Type

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_Append Update	"a+"	Open file for reading and writing, to the end of existing file, or creating a new file; whole file can be read, but writing only appended

OUTPUTS: file_handle-used to manipulate files with other "C" library stream I/O routines

RETURNS:

Table 6-38. PGS_IO_Gen_Temp_Open Returns

Return	Description
PGS_S_SUCCESS	Success
PGSIO_W_GEN_ACCESS_MODIFIED	Illegal attempt to open existing file for access mode PGSd_IO_Gen_Write or PGSd_IO_Gen_Trunc; Access mode reset to PGSd_IO_Gen_AppendUpdate
PGSIO_W_GEN_NEW_FILE	File expected, but was missing; new file created
PGSIO_W_GEN_DURATION_NOMOD	Attempt to alter existing intermediate duration attribute ignored
PGS_E_UNIX	UNIX system error
PGSIO_E_GEN_OPENMODE	Invalid access mode
PGSIO_E_GEN_REFERENCE_FAILURE	Can not find physical file name with logical ID in $PGS_PC_INFO_FILE
PGSIO_E_GEN_BAD_FILE_DURATION	Invalid file duration
PGSIO_E_GEN_FILE_NOEXIST	No entry for logical ID $PGS_PC_INFO_FILE
PGSIO_E_GEN_CREATE_FAILURE	Error creating new file entry in $PGS_PC_INFO_FILE
PGSIO_E_GEN_NO_TEMP_NAME	Failed to create temporary filename
PGSIO_E_GEN_BAD_ENVIRONMENT	Bad environment detected for I/O path **...**

"Existing file" means that an entry for the file exists in $PGS_PC_INFO_FILE.

(NOTE: the above are short descriptions only; full text of messages appears in files $PGSMSG/*.t . Descriptions may change in future releases depending on external ECS design.)

EXAMPLE:

```
// This example illustrates how to create an Intermediate
   File //

PGSt_SMF_status         returnStatus;
PGSt_PC_Logical         logical;
PGSt_IO_Gen_FileHandle  *handle;

#define     INTER_1B 101

returnStatus =
PGS_IO_Gen_Temp_Open(PGSd_IO_Gen_Endurance,INTER_1B,
                     PGSd_IO_Gen_Write,&handle );
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES: This function will support most POSIX modes of fopen; the only exception being truncate mode (w+).

Logical identifiers used for files may NOT be duplicated.

Existing files will NOT be overwritten by calling this function in mode PGSd_IO_Gen_Write. Instead, they will be opened in PGSd_IO_Gen_AppendUpdate mode; a warning will be issued signifying that this is the case. Warnings will also be issued in the event that a non-existent file is opened in modes other than explicit write (i.e., PGSd_IO_Gen_Append, or PGSd_IO_Gen_AppendUpdate).

By using this tool, the user understands that a Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this Temporary file prior to PGE termination, it will be purged by the Science Data Processing Segment (SDPS) system during normal cleanup operations. If the user requires a more static instance of a file, one that will exist beyond normal PGE termination, that user may elect to create an Intermediate file instead by specifying some persistence value (currently, PGSd_IO_Gen_Endurance is the only value recognized); note that this value is only valid for the initial creation of a file and will not be applied to subsequent accesses of the same file.

The following table gives proper use of the *file_duration* input variable:

Table 6-39. Proper Use of Persistence Values

File Type & Access	Duration Factors
TEMPORARY	
Creation	PGSd_IO_Gen_NoEndurance
Repeated Access	NULL
INTERMEDIATE	
Creation	PGSd_IO_Gen_Endurance
Repeated Access	NULL

FILE CHARACTERISTICS

All files created by this function have the following form:

[label][global-network-IP-address][process-id][date][time]

where:

label : SDP Toolkit Process Control -> pc

global-network-IP-address: complete IP address iii.iii.iii.iii -> iiiiiiiiiiii

(0's padded to maintain triplet groupings)

process-id : process identifier of current executable -> pppppp

date : days from beginning of year & the year -> dddyy

time : time from midnight local time -> hhmmss

Table 6-40. Temporary File Name Definition

Field	Description	Format
label	SDP Toolkit Process Control	"pc"
production-run-id	numeric identifier from 1 to 8 places	rrrrrrrr
local-network-IP-address	local portion of Internet protocol (IP) address uuu.vvv.ww.xx	vvvwwxx
process-id	UNIX identifier for current process	pppppp
date	# days from beginning of year, and the year	dddyy
time	time from midnight local time	hhmmss

Reference names returned by this function have the following form:

[label][global-network-IP-address][process-id][date][time]

where:

label : SDP Toolkit Process Control -> pc

global-network-IP-address: complete IP address iii.iii.iii.iii -> iiiiiiiiiiii

(0's padded to maintain triplet groupings)

process-id : process identifier of current executable -> pppppp

date : days from beginning of year & the year -> dddyy

time : time from midnight local time -> hhmmss

or 'pciiiiiiiiiiiipppppppddddddttttttt'

ex. pc198118192017010280003951040 34

pc 198118192017 010280 00395 104034

| | | | | |

(pc) label_____| | | | |

(i) full-network-IP-address _____| | | |

(p) process-id_____| | |

(d) date_____| |

(t) time_____|

All temporary and intermediate files generated by this tool are uniquewithin the global ECS community. Also, all file names are NOW exactly 31 characters in length; this should help with the diagnosis of suspect temporary files (i.e., check the length first).

NOTE: Users should NOT put entries in the TEMP or INTERMEDIATE OUTPUT sections. The Toolkit will do this.

The behavior of the Toolkit routine PGS_IO_Gen_Temp_Open() of not allowing file truncations was part of the original design (this is a "feature" not a bug). I believe the idea was that NO data should be destroyed (not even intermediate/temporary data). The actual solution for truncation (to fit the original design) is to delete the temporary files a routine uses when it exits the routine. This is done with the Toolkit call PGS_IO_Gen_Temp_Delete(). This will allow the reuse of the same logical ID to create a temporary file each time the routine is called. The general usage is: invoke PGS_IO_Gen_Temp_Open() to open the temporay file do processing making use of temporary file close the temporary file using PGS_IO_Gen_Close() delete the temporary file using PGS_IO_Gen_Temp_Delete() repeat as necessary

REQUIREMENTS: PGSTK-0530, PGSTK-0531

Open a Temporary/Intermediate File (FORTRAN Version)

NAME: **PGS_IO_Gen_Temp_OpenF( )**

SYNOPSIS:

C: (not applicable)

FORTRAN:
```
INCLUDE    'PGS_SMF.f'
INCLUDE    'PGS_PC_9.f'
INCLUDE    'PGS_IO.f'
INCLUDE    'PGS_IO_1.f'

integer function pgs_io_gen_temp_openf(file_duration, file_logical,
                                       file_access, record_length, file_handle)
integer file_duration
integer file_logical
integer file_access
integer record_length
integer file_handle
```

DESCRIPTION: Upon a successful call, this function will return a logical unit number for use with FORTRAN READ and WRITE statements. This is returned to the user via the parameter file_handle. The user provides the logical file identifier that internally gets mapped to the associated physical file. The user also provides the file duration parameter, to specify whether the file being opened is to be temporary or intermediate.

INPUTS: file_duration-specifies how long file will last:

Table 6-41. File Duration

PGS-defined value	Description
PGSd_IO_Gen_Endurance	intermediate file
PGSd_IO_Gen_NoEndurance	temporary file

file_logical-User defined logical file identifier

file_access-type of access granted to opened file:

Table 6-42. File Access Type

PGS FORTRAN Access Mode	Rd/Wr/Update/Append	FORTRAN 77/90 'access='	FORTRAN 77/90 '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
F90 SPECIFIC			
PGSd_IO_Gen_AseqFrm	Append	Sequential	Formatted
PGSd_IO_Gen_AseqUnf	Append	Sequential	Unformatted

record_length-record length for direct access IO:
mandatory for direct access (minimum value = 1)
ignored otherwise

F90 SPECIFIC must be greater than or equal to 0 for sequential access; if 0, file is opened with default record length

OUTPUTS: file_handle-used to manipulate files with READ and WRITE

RETURNS:

Table 6-43. PGS_IO_Gen_Temp_OpenF Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSIO_E_NO_FREE_LUN	All logical unit numbers are in use
PGSIO_W_GEN_ACCESS_MODIFIED	The access mode has been modified
PGSIO_E_GEN_OPENMODE	Illegal open mode was specified
PGSIO_E_GEN_OPEN_OLD	Attempt to open with STATUS=OLD failed
PGSIO_E_GEN_OPEN_NEW	Attempt to open with STATUS=NEW failed
PGSIO_E_GEN_OPEN_RECL	Invalid record length specified
PGSIO_W_GEN_OLD_FILE	File exists: changing access to update
PGSIO_W_GEN_NEW_FILE	File not found, created new one
PGSIO_W_GEN_DURATION_NOMOD	Illegal attempt to modify file duration
PGSIO_E_GEN_REFERENCE_FAILURE	Can't do Temporary file reference
PGSIO_E_GEN_BAD_FILE_DURATION	Illegal file duration value
PGSIO_E_GEN_FILE_NOEXIST	File not found, cannot create
PGSIO_E_GEN_CREATE_FAILURE	Unable to create new file
PGSIO_E_GEN_NO_TEMP_NAME	New name could not be generated

EXAMPLE:

```
integer returnstatus
integer file_duration
integer file_logical
integer file_access
integer record_length
integer file_handle

file_duration     = PGSd_IO_Gen_NoEndurance
file_logical      = 101
file_access       = PGSd_IO_Gen_WDirUnf
record_length     = 1

returnstatus = PGS_IO_Gen_Temp_OpenF(file_duration,
                                     file_logical,
                                     file_access,
                                     record_length,
                                     file_handle)

if (returnstatus .NE. PGS_S_SUCCESS) then

C   goto 1000
endif
.
.
.

100 <error handling goes here>
```

NOTES:

Logical identifiers used for Temporary and Intermediate files may NOT be duplicated. Existing files will NOT be overwritten by calling this function in any of the write modes. Instead, they will be opened in the corresponding update mode; a warning will be issued signifying that this is the case. Warnings will also be issued in the event that a nonexistent file is opened in modes other than explicit write.

By using this tool, the user understands that a Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this file prior to PGE termination, it will be purged by the PGS system during normal cleanup operations. If the user requires a more static instance of a file, one that will exist beyond normal PGE termination, that user may elect to create an Intermediate file instead by specifying some persistence value (currently, PGSd_IO_Gen_Endurance is the only value recognized); note that this value is only valid for the initial creation of a file and will not be applied to subsequent accesses of the same file.

In order to insure that generated temporary file names are unique for the same host, a delay factor of 1 millisecond is imposed during the name creation process.

Due to the nature of FORTRAN IO, it is possible to write a file opened for reading as well as read a file opened for writing. The matching of access mode to IO statement cannot be enforced by the tool. This is up to the user.

Once a file has been opened with this tool, it must be closed with a call to PGS_IO_Gen_CloseF before being re-opened. Failure to do this will result in undefined behavior.

REQUIREMENTS: PGSTK-0530, PGSTK-0531

Delete a Temporary File

NAME: **PGS_IO_Gen_Temp_Delete( )**

SYNOPSIS:

C:

```
#include <PGS_IO.h>

PGSt_SMF_status
PGS_IO_Gen_Temp_Delete(
        PGSt_PC_Logical     file_logical);
```

FORTRAN:

```
INCLUDE     'PGS_SMF.f'
INCLUDE     'PGS_PC_9.f'
INCLUDE     'PGS_IO_1.f'

integer pgs_io_gen_temp_delete(
        integer file_logical)
```

DESCRIPTION: Upon a successful call, this function will "effectively" delete the Temporary file currently referenced by the specified logical identifier. (See NOTES.) Future references to this logical identifier will no longer provide access to a file until such time as a new temporary file is created with the same logical identifier.

INPUTS: file_logical-User defined logical file identifier

OUTPUTS: None

RETURNS:

PGS_S_SUCCESS
PGSIO_E_GEN_REFERENCE_FAILURE
PGSIO_E_GEN_FILE_NODEL
PGSIO_W_GEN_FILE_NOT_FOUND

EXAMPLE:

```
PGSt_SMF_status   ret_val;
PGSt_PC_Logical   logical;

#define     INTER_1B 101

ret_val = PGS_IO_Gen_Temp_Delete( INTER_1B );
if (ret_val != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
    .
    .
    .
EXCEPTION:
```

NOTES: The actual deletion of Temporary files is not carried-out until after the completion of the PGE run. Instead, these files are marked as deleted through the Process Control mechanism. This allows for the preservation of all Temporary files generated during a PGE run, to facilitate error tracking/debugging following a failed run of a PGE. This in no way prevents the creation of a new temporary file using the same logical identifier as one previously deleted.

Unlike all other IO_Gen tools, this function has a FORTRAN binding to C. There is no separate FORTRAN version.

Logical identifiers used for Temporary and Intermediate files may NOT be duplicated.

By using this tool, the user understands that a truly Temporary file may only exist for the duration of a PGE. Whether or not the user deletes this file prior to PGE termination, it will be purged by the Science Data Processing System (SDPS) system during normal cleanup operations.

REQUIREMENTS: PGSTK-0520

6.2.2 Error/Status Reporting (SMF Tools)

To detect and report an error and status conditions in a consistent manner across the ECS, standardized status messages and status codes must first be established. The method used to institute these message/code pairs is by way of the 'smfcompile' utility. But first, users will need to create Status Message Files (SMFs) to contain their custom status messages and corresponding status identifiers. These identifiers take the form of user defined mnemonics that visually convey the essence of the status message. The user will make direct use of these mnemonics in their software when testing for status conditions and when interfacing with the SMF Toolkit functions. Once an SMF is completed, the smfcompile utility is run in order to bind the status messages and mnemonics with integral status codes. This process facilitates the runtime access of all status messages and provides for the referencing of status mnemonics within the user's code.

The status codes generated by the 'smfcompile' utility are guaranteed to be unique across the entire SDPS system to ensure that there will be no ambiguous status conditions, in the event that code from different Science Computing Facilities (SCFs) is merged into a single executable and/or PGE. This uniqueness is possible because "seed" values, which are different for every SMF, are used in the generation of the status codes. Typically, many SMF files will be created in the course of software development; therefore many seed numbers will be required. However, it is important to note that valid seed numbers can only be obtained from the Toolkit development team (landover_PGSTLKIT@raytheon.com). Any attempt to produce SMFs from "home-grown" seed values may result in the SMFs being unusable at integration & test time.

The SDP Toolkit routines actually contain their own collection of status codes and associated status messages for describing the state of each Toolkit function. Users of the Toolkit functions should examine the return values of each tool before performing any other action. To inform a calling unit (user's software) about the exit state of a called Toolkit routine, each Toolkit function sets a status message and assigns a status code to the return value. On the basis of its interpretation of this return value, the calling unit may elect to perform some error handling. As part of this procedure, the user should either propagate the existing status code up through their calling hierarchy, or set a status code and message to represent the outcome of any local error handling attempt.

Upon detection of an error state, users are advised to report on the existing error prior to performing an error handling procedure. The content of these reports might include the following: a user-defined message string to convey the nature of the status condition, a user-defined action string to indicate the next operation to be performed in response to the status condition, and a system defined string that uniquely identifies the environment in which the status condition occurred. However, this is merely a suggestion; the user is free to define the content of the status reports to satisfy their own requirements. The method for reporting this information will involve the generation of a report from the information just described and the subsequent transmission of that report to the appropriate destination(s).

Once software development has been completed, all the Status Message Files (SMFs) created to support that development will be delivered to the DAAC along with the developed PGE(s). The

Toolkit SMFs will be delivered to the DAACs along with the Toolkit library, just as they were delivered to the SCFs.

The tools provided here allow for the propagation of status information within a PGE executable to facilitate a user's error handling process. They also provide the means to communicate status and error information to various monitoring authorities and event logs. Additionally, there is a tool that enables the user to specify, a priori, the action to be taken in the wake of a fatal arithmetic event. This mechanism will allow the user to take their own corrective measures to control an event that is terminal by default. Note that all other event conditions fall under the purview of system processing and are thereby controlled by the governing SDPS software.

Several new features have been incorporated into these tools for Toolkit 5 in order to improve their efficiency. One of those features allows for the buffering of individual status messages up to some user defined runtime limit. This should greatly reduce the amount of I/O required to access these messages. As a process proceeds to completion, new status messages are buffered as older, less used status messages become unbuffered. The goal here is to only access status messages from their runtime file when they are being referenced for the first time. The actual observed improvement will depend on the degree to which a process' status messages are localized (i.e., A particular status message should ideally only be referenced within a short body of code.) and the buffer size, which is set by the user. Another feature reduces the number of replicated status messages that can appear in the status log file. This is accomplished by "compressing" duplicate messages into a count of such messages. This feature should significantly reduce the size of the status log file and contribute to its general readability.

Please refer to Appendix B for guidance on the creation of Status Message Files and for examples of SMFs and explicit SMF Toolkit usage.

6.2.2.1 Log File Output Control

Several new features have been added to the Toolkit to allow greater control of message logging. The behavior of these features is controlled via entries in the Process Control File (PCF). Note that the use of some or all of these features may be strictly controlled at the DAACs.

6.2.2.1.1 Logging Control

PCF entry:
10114|Logging Control; 0=disable logging, 1=enable logging|1

This may be used to disable logging altogether. If logging is disabled NO message will output to any log files (although a small header will still be written to the log files indicating that for this PGE logging has been disabled). The default state is for logging to be enabled.

6.2.2.1.2 Trace Control

PCF entry:
10115|Trace Control; 0=no trace, 1=error trace, 2=full trace|0

This may be used to specify the trace level for message logging. Tracing is a feature made possible by the addition of two new SMF tools: PGS_SMF_Begin and PGS_SMF_End (see the respective entries in 6.2.2.2 Status Reporting Tools). Users may include these tools at the beginning and ending of their functions (respectively) to signal to the SMF system when each user defined function is entered and exited. Three levels of tracing are possible:

No Tracing

This is the default state. No information concerning the entering or exiting of functions is recorded to the log files. No information concerning the path of a function call is recorded to the log files.

Example Log Entry:
func4():PGSTD_W_PRED_LEAPS:27652
predicted value of TAI-UTC used (actual value unavailable)

Error Tracing

If error tracing is enabled, information concerning the path of a function call is recorded to the log files any time a status message is logged to a log file. This is useful in determining where in a chain of function calls an error occurred. No information concerning the entering or exiting of functions is recorded in this state.

Example Log Entry:
main():
  func1():
    func2():
      func3():
        func4():PGSTD_W_PRED_LEAPS:27652
        predicted value of TAI-UTC used (actual value unavailable)

Full Tracing

If full tracing is enabled, a message will be written to the log files each time a function is entered and exited (only those user functions with the PGS_SMF_Begin/End calls, see above). Indenting will also be done to show the path of each function call.

Example Log Entry:

PGS_SMF_Begin: main()

  PGS_SMF_Begin: func1()

    PGS_SMF_Begin: func2()

      PGS_SMF_Begin: func3()

        PGS_SMF_Begin: func4()

          func4():PGSTD_W_PRED_LEAPS:27652
          predicted value of TAI-UTC used (actual value unavailable)

PGS_SMF_End: func4()

PGS_SMF_End: func3()

PGS_SMF_End: func2()

PGS_SMF_End: func1()

PGS_SMF_End: main()

6.2.2.1.3 Process ID Logging

PCF entry:
10116|Process ID logging; 0=don't log PID, 1=log PID|0

This may be used to enable the tagging of log file entries with the process ID of the process from which the entry came. This is useful for PGEs that run concurrent processes which will all be writing to a single log file simultaneously. If process ID logging is enabled, each log entry will be tagged with the process ID of the process making the entry. This can facilitate in post-processing a log file.

Example Log Entry:
func4():PGSTD_W_PRED_LEAPS:27652 (PID=2710)
predicted value of TAI-UTC used (actual value unavailable)

6.2.2.1.4 Status Level Control

PCF entry:
10117|Disabled status level list (e.g., W S F)|<status level list>

This may be used to disable the logging of status codes of specific severity levels. A list of levels to be disabled should be substituted for <status level list> (e.g.: N M U). No message of a status level indicated in the list will be recorded to any log file (see Appendix B for details on status message levels). The default state is to enable logging for all status levels.

6.2.2.1.5 Status Message Seed Control

PCF entry:
10118|Disabled seed list|<status code seed list>

This may be used to disable the logging of status codes generated from specific seed values. A list of seed values, the status codes derived from which should be disabled, should be substituted for <status code seed list> (e.g.: 3 5). No message derived from a seed value indicated in the list will be recorded to any log file (see Appendix B for details on status message seed values). The default state is to enable logging for all seed values.

6.2.2.1.6 Individual Status Code Control

PCF entry:
10119|Disabled status code list|<status code list>

This may be used to disable the logging of specific status codes. A list of status code mnemonics and/or numeric status codes should be substituted for <status code list> (e.g.: PGSTD_M_ASCII_TIME_FMT_B 678954). Note that only Toolkit status codes can be disabled by using mnemonics. To disable a user generated status code a numeric status code must be used. No messages whose status codes or mnemonics are included in the list will be recorded to any log file. The default state is to enable logging for all status codes.

6.2.2.1.7 Generating Runtime E-Mail Messages

A PGE may be configured to automatically generate and send e-mail message during runtime when specific user defined status codes are logged. This is done by assigning an e-mail action to a given user defined status code.

An e-mail action is an SMF code with the special status level of "C" and a mnemonic that begins with the characters "PGSEMAIL" (the rest of the mnemonic may contain any other valid mnemonic characters), for example:

PGS_*C_PGSEMAIL*_SEND_EMAIL
ASTER_*C_PGSEMAIL*_ALERT
MODIS_*C_PGSEMAIL*_ERROR

An e-mail message will be generated anytime a user defined status code with an associated e-mail action is logged via the SMF logging routines. The contents (body) of these messages will be the text (message) associated with the user defined status code. The subject of these messages will be the mnemonic associated with the user defined status code. The list of recipients is defined in the e-mail action definition.

Example:
In a user defined status message file the following status code mnemonic label and e-mail action mnemonic label have been defined (the e-mail action is associated with the status code via the "::" syntax):

```
MODIS_E_PGE_INIT_FAILED   The PGE failed to initialize.
                          ::MODIS_C_PGSEMAIL_NOTIFY
MODIS_C_PGSEMAIL_NOTIFY   john@modis.org, sue@modis.org
```

The following lines appear in a C source code file:

```
    returnStatus = initializePGE();
    if (returnStatus == MODIS_E_PGE_INIT_FAILED)
    {
        PGS_SMF_SetStaticMsg(returnStatus, "main()");
```

```
        exit(1);
    }
```

At runtime, if the returned status code from the function initializePGE() has the value defined by MODIS_E_PGE_INIT_FAILED, this status is logged via the SMF function PGS_SMF_SetStaticMsg(), and because this status code has an e-mail action associated with it, an e-mail message will be generated.

The e-mail message will be sent to: sue@modis.org and john@modis.org
The subject field of the e-mail message will be: MODIS_E_PGE_INIT_FAILED
The text of the e-mail message will be: The PGE failed to initialize.

Note:
This functionality will be disabled at the DAACs.

6.2.2.2 Status Reporting Tools

Get Toolkit Version

NAME: **PGS_SMF_GetToolkitVersion()**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

void
PGS_SMF_GetToolkitVersion(
        char  version[21]);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_gettoolkitversion(
        character*20 version)
```

DESCRIPTION: This function returns a string describing the current version of the Toolkit.

INPUTS: None

OUTPUTS: version - character string describing the current version of the Toolkit

RETURNS: None

EXAMPLES:

C:

```
char version[21];
PGS_SMF_GetToolkitVersion(version);
```

FORTRAN:

```
character*20
call pgs_smf_gettoolkitversion(version)
```

NOTES: User must allocate enough memory to hold the Toolkit version string.This function does not allocate any memory for the user.

REQUIREMENTS:

Set UNIX Status Message

NAME: **PGS_SMF_SetUNIXMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetUNIXMsg(
        PGSt_integer  unix_errcode,
        char          *msg,
        char          *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_setunixmsg(unix_errcode,msg,funcname)
        integer         unix_errcode
        character*240   msg
        character*32            funcname
```

DESCRIPTION: This tool provides the means to retain UNIX error messages for later retrieval. Additionally, the user has the flexibility to append a user defined message to a UNIX message for further clarity.

INPUTS: unix_errcode-the error code set by C library; UNIX system calls; and POSIX FORTRAN calls, i.e., the value stored in C 'errno' and Fortune 'IERROR'

msg-user defined status message string

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-44. PGS_SMF_SetUNIXMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_LOGFILE	Error opening status, report or user files
PGSSMF_E_UNDEFINED_UNIXERRNO	Undefined UNIX error
PGSSMF_E_MSG_TOOLONG	Message length exceeded

EXAMPLES:

C:

This example uses the ‘popen()’ C library routine merely to illustrate how the SMF tool PGS_SMF_SetUNIXMsg() might be used to preserve the UNIX error condition. Note that ‘popen()’ is not part of the POSIX standard and therefore should not be used within the science software.

```
PGSt_SMF_status Get_Listing()
{
   FILE                *stream;
   char                buffer[101];
   char                directoryEntry[101];
   PGSt_SMF_status     returnStatus = PGS_S_SUCCESS;

   if (stream = popen(“ls”,”r”) != NULL)
   {
      while (fgets(buffer,100,stream) != NULL)
         {
            scanf(buffer,”%s”,directoryEntry);
         }
   }
   else
   {
       PGS_SMF_SetUNIXMsg(errno,NULL,”Get_Listing()”);
       pclose(stream);
       returnStatus = PGS_E_UNIX;
   }
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_setunixmsg
character*1 chr
integer     ierror

PXFFGETC(IPXFCONST(“STDIN_UNIT”),chr,ierror)
IF (ierror .NE. 0) THEN
   pgs_smf_Setunixmsg(ierror,’PXFFGETC() error
occured’,’Get_Listing()’)
ENDIF
```

NOTES:

The parameter “funcname” can be passed in as NULL if you do not wish to record the routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes. Likewise, the parameter “msg” can be NULL unless you wish to have an

additional message appended to the system defined UNIX message. The static variable ‘errno’ has been declared in ‘PGS_SMF.h’. Since UNIX treats errno as a static parameter, the user will have to save the value returned from the critical call unless the call to ‘PGS_SMF_SetUNIXMsg()’ is made immediately. If unix_errno is not a valid constant, the static buffer will be updated with the appropriate error message.

This tool is primarily intended for users of the C programming language. However, we believe that this functionality will support users of the POSIX FORTRAN language as well. Please refer to POSIX FORTRAN 77 IEEE Std 1003.9-1992 on page 14, Section 2.4 (Error Numbers) for information regarding POSIX FORTRAN’s implementation of standard error return values.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0632, PGSTK-0650

Set Static Status Message

NAME: **PGS_SMF_SetStaticMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetStaticMsg(
        PGSt_SMF_code       code,
        char                *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_setstaticmsg(code,funcname)
        integer code
        character*32    funcname
```

DESCRIPTION: This tool will provide the means to set a pre-defined error/status message in response to the outcome of some segment of processing.

INPUTS: code-mnemonic error/status code generated by message compiler (see "smfcompile")

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-45. PGS_SMF_SetStaticMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error message
PGSSMF_E_LOGFILE	Error opening status, report or user files
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
returnStatus =
      PGS_SMF_SetStaticMsg(PGSSMF_E_UNDEFINED_UNIXERROR,
      "My_Function()");
```

FORTRAN:

```
implicit none

integer     returnstatus
integer     pgs_smf_setstaticMsg
returnstatus =
```

```
pgs_smf_setstaticMsg(PGSSMF_E_UNDEFINED_UNIXERROR,
'my_function()')
```

NOTES: The parameter “funcname” can be passed in as NULL if you do not wish to record that routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0650

Set Dynamic Status Message

NAME: **PGS_SMF_SetDynamicMsg( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SetDynamicMsg(
        PGSt_SMF_code     code,
        char              *msg,
        char              *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_setdynamicmsg(code,msg,funcname)
        integer           code
        character*240     msg
        character*32      funcname
```

DESCRIPTION: This tool will provide the means to set a runtime specific status message, for a particular status code, in response to the outcome of come segment of processing.

INPUTS: code-mnemonic error/status code generated by message compiler

msg-message string to be saved into the static buffer

funcname-function where the status condition occurred

OUTPUTS: None

RETURNS:

Table 6-46. PGS_SMF_SetDynamicMsg Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_LOGFILE	Error opening status, report or user files

EXAMPLES:

C: Having defined a mnemonic code in the SMF file:

```
INSTR_E_BAD_CALIBRATION Calibration value %7.2f
                        is not within tolerance
```

We would like to insert the calibration factor into the message template during processing, since the value is not fixed prior to runtime. The message that would be set in the status buffer would then appear as:

```
'Calibration value 356.23 is not within tolerance'
```

```
PGSt_SMF_status   returnStatus;
PGSt_SMF_code     code;
char              msg[PGS_SMF_MAX_MSG_SIZE];
char              buf[PGS_SMF_MAX_MSGBUF_SIZE];
float             calibration_factor = 356.23;

calibration_factor = Get_Instrument_Calibration( NIGHT );
/# value of 356.23 returned #/

returnStatus =
PGS_SMF_GetMsgByCode(INSTR_E_BAD_CALIBRATION,msg);
sprintf(buf,msg,calibration_factor);

PGS_SMF_SetDynamicMsg(INSTR_E_BAD_CALIBRATION,buf,Level1A_In
itialization()")
```

FORTRAN: Having defined a mnemonic code in the SMF file:

```
INSTR_E_BAD_CALIBRATION Calibration value is not
                  within tolerance ->
```

We would like to insert the calibration factor to the end of the message template during processing, since the value is not fixed prior to runtime. The message that would be set in the status buffer would then appear as:

```
'Calibration value is not within tolerance -> 356.23'
```

```
implicit none

integer           pgs_smf_getmsgbycode
integer           pgs_smf_setdynamicmsg
integer           returnstatus
character*240     msg
character*480     buf
real              calibration_factor
integer           msglen
character*8       coeff_str

calibration_factor = get_instrument_calibration( NIGHT )

C   value of 356.23 returned
                  returnstatus = pgs_smf_getmsgbycode(
                    INSTR_E_BAD_CODE,msg)
```

```
write( coeff_str,'(F7.2)') calibration_factor
msglen = len( msg)
buf = msg(1:msglen)//coeff_str

pgs_smf_setdynamicmsg( INSTR_E_BAD_CALIBRATION, buf,
  'level1A_initialization' );
```

NOTES: Note that you can have the flexibility of associating any dynamic message string to the defined mnemonic code via this routine.

This tool can be used in various situations. For instance the user might want to concatenate some message strings together and assign the resultant string to an existing mnemonic code, so that this message can be passed forward to another module for further processing. Alternatively it can be used to embed runtime variables in the defined message template before saving this message string to the static message buffer.

The parameter "funcname" can be passed in as NULL if you do not wish to record the routine that noted this error. However, it is strongly recommended that you pass the routine name for tracking purposes.

The parameter "msg" can be passed in as NULL. If you do, no message is associated with the mnemonic code.

Refer to utility "smfcompile" for additional information on the format of the message compiler.

REQUIREMENTS: PGSTK-0582, PGSTK-0600, PGSTK-0650

Get Status Message by Code

NAME: **PGS_SMF_GetMsgByCode( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetMsgByCode(
        PGSt_SMF_code       code,
        char                msg[]);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_getmsgbycode(code,msg)
        integer         code
        character*240   msg
```

DESCRIPTION: This tool will provide the means to retrieve the message string that is associated with a specific status code in the Status Message Files.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: msg-user pre-defined message string

RETURNS:

Table 6-47. PGS_SMF_GetMsgByCode Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES: See example for PGS_SMF_SetDynamicMsg( ).

NOTES: This tool provides a simple Status Message File (SMF) lookup function. It should be used primarily for retrieving messages that contain C-style formatting tokens to facilitate the replacement of those tokens with runtime data.

REQUIREMENTS: PGSTK-0580, PGSTK-0650

Get Status Message

NAME: **PGS_SMF_GetMsg( )**

SYNOPSIS

C:

```
#include <PGS_SMF.h>

void
PGS_SMF_GetMsg(
        PGSt_SMF_code     *code,
        char              mnemonic[],
        char              msg[]);
```

FORTRAN:

```
call pgs_smf_getmsg(code,mnemonic,msg)
        integer           code
        character*32      mnemonic
        character*480     msg
```

DESCRIPTION: This tool will provide the means to retrieve status information from the static buffer, for use when reporting on specific status conditions.

INPUTS: None

OUTPUTS: mnemonic-previously set mnemonic error/status string

msg-previously set message string

RETURNS: None

EXAMPLES: See example for PGS_SMF_SetDynamicMsg( ).

NOTES: Until a call is made which sets status information into the buffer, none exists. Therefore, first time calls to this function may return the following for each of the arguments: code=0, mnemonic="", and msg="".

A call to any of the PGS_SMF_Set*( ) functions will load status information into the static buffer. To ensure that the caller of your function can receive the intended information, calls to the PGS_SMF_Set*( ) functions should be performed just prior to returning control back to the caller.

To ensure that the status information received pertains to the status condition set during the last function call, it is imperative that the user invoke this function immediately upon gaining control back from the function that set the status information.

REQUIREMENTS: PGSTK-0580, PGSTK-0650

Get Action Message by Code

NAME: **PGS_SMF_GetActionByCode( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetActionByCode(
        PGSt_SMF_code       code,
        char                action[]);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_getactionbycode(code,action)
        integer         code
        character*240   action
```

DESCRIPTION: This tool will provide the means to retrieve an action string corresponding to a specific mnemonic code.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: action-associated action string

RETURNS:

Table 6-48. PGS_SMF_GetActionByCode Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_W_NOACTION	No action defined
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char  action[PGS_SMF_MAX_ACT_SIZE];

returnStatus =
PGS_SMF_GetActionByCode(PGSSMF_E_UNDEFINED_UNIXERROR,
      action);
if (returnStatus != PGS_S_SUCCESS)
{
   /# could not retrieve action message #/
}
```

```
else
{
   /# generate a status report and indicate action to be
      taken #/
}
```

FORTRAN:

```
implicit none

integer           pgs_smf_getactionbycode
integer           returnstatus
character*240     action
    returnstatus = pgs_smf_getactionbycode(
      PGSSMF_E_UNDEFINED_UNIXERROR, action );
    IF (returnstatus .NE. PGS_S_SUCCESS) THEN

C   could not retrieve action message
    ELSE

C   generate status report and indicate action to be taken
    ENDIF
```

NOTES: This routine will not return any associated action string if the creator of the status code did not associate an action label when creating the Status Message File entry for that status code. If this is the case, the resulting parameter is action[0] = '\0'. Refer to the available documentation for the 'smfcompile' utility for additional information on how to define and attach action messages to status code entries.

REQUIREMENTS: PGSTK-0591, PGSTK-0650

Create Message Tag

NAME: **PGS_SMF_CreateMsgTag( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_CreateMsgTag(
        char    systemTag[]);
```

FORTRAN:

```
integer function pgs_smf_createmsgtag(systemtag)
        char*60         systemtag
```

DESCRIPTION: The tool described here allows the user to generate a runtime specific character string that may be useful for tagging important items of data. The string contains system defined identifiers that, when combined, can be useful for stamping non-product specific data for system traceability.

INPUTS: None

OUTPUTS: systemTag-system defined message string

RETURNS:

Table 6-49. PGS_SMF_CreateMsgTag Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_W_NO_CONSTRUCT_TAG	No information to construct message tag
PGSSMF_E_BAD_REFERENCE	Bad reference

EXAMPLES:

C:

```
char  systemTag[PGSd_SMF_TAG_LENGTH_MAX];
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_CreateMsgTag(systemTag);
if (returnStatus == PGS_S_SUCCESS)
{
   /# create message tag successful #/
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_createmsgtag
char*60     systemtag
integer     returnstatus
```

```
                returnstatus = pgs_smf_createmsgtag(systemtag)
                    IF (returnstatus .EQ. PGS_S_SUCCESS) THEN

C   create message tag successful
                    ENDIF
```

NOTES: Currently, the only system identifiers used to create the message tag are:

the Science Software Configuration ID,

and the Production Run ID.

IMPORTANT TOOLKIT NOTES

The logical parameter identifiers, which are implicitly defined by the PC tools, are internally mapped to an associated physical parameter through the Process Control mechanism. Therefore before this tool can be used, a Process Control Table MUST be created and properly filled out. In addition, the following environment variables must be set to ensure proper operation:

PGS_PC_INFO_FILE path to process control file

REQUIREMENTS: PGSTK-0610

Get Instrument Name

NAME: **PGS_SMF_GetInstrName( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GetInstrName(
        PGSt_SMF_code       code,
        char                instr[]);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_getinstrname(code,instr)
        integer code
        character*10    instr
```

DESCRIPTION: This tool may be used to retrieve the instrument name from a given error/status code.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: instr-corresponding instrument name as it appears in the message text file after the token %INSTR.

RETURNS:

Table 6-50. PGS_SMF_GetInstrName Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status         returnStatus;
char  instr[PGS_SMF_MAX_INSTR_SIZE];

returnStatus = PGS_SMF_GetInstrName(MODIS_E_BAD_CALIBRATION
,instr);
if (returnStatus == PGS_S_SUCCESS)
{
   /# record instrument that generated instrument condition
#/
}
```

FORTRAN:

```
implicit none

integer           pgs_smf_getinstrname
integer           returnstatus
character*10      instr

returnstatus = pgs_smf_getinstrname(
      MODIS_E_BAD_CALIBRATION, instr )
IF (returnstatus .EQ. PGS_S_SUCCESS) THEN

C   record instrument which generated status condition
ENDIF
```

NOTES: This function may be useful for programs which link in libraries created by cooperating instrument teams, and where the need to distinguish the status conditions associated with each instrument team arises.

REQUIREMENTS: PGSTK-0620, PGSTK-0650

Generate Status Report

NAME: **PGS_SMF_GenerateStatusReport( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_GenerateStatusReport(
        char     *report);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_generatestatusreport(report)
        char*1024       report
```

DESCRIPTION: This tool provides the method for the user to create status reports for use by Science Computing Facility personnel. Each call to this procedure causes the user defined report to be appended to the status report log.

INPUTS: report-user report generated text

OUTPUTS: None

RETURNS:

Table 6-51. PGS_SMF_GenerateStatusReport Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_LOGFILE	Error opening status, report or user files

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;

returnStatus = PGS_SMF_GenerateStatusReport("Write it into
status report file");
if (returnStatus == PGS_S_SUCCESS)
{
   /# write to status report successful #/
}
```

FORTRAN:

```
implicit none

integer     pgs_smf_cgeneratestatusreport
integer     returnStatus
```

```
                returnStatus = pgs_smf_cgeneratestatusreport("Write it into
                      status report file")
                IF (returnStatus .EQ. PGS_S_SUCCESS) THEN
C   write to status report successful
                ENDIF
```

NOTES: The system defined message tag will automatically be added to the user-provided report.

IMPORTANT TOOLKIT NOTES

The logical file identifier (PGSd_SMF_LOGICAL_LOGSTATUS), which is implicitly used by this tool, is internally mapped to an associated physical file through the Process Control mechanism. Therefore before this tool can be used, a Process Control Table MUST be created and properly filled out. In addition, the following environment variables must be set to ensure proper operation:

Table 6-52. Environment Variables

Variable	Path
PGS_PC_INFO_FILE	path to process control file

REQUIREMENTS: PGSTK-0650

NAME: **PGS_SMF_SendRuntimeData( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_SendRuntimeData(
        PGSt_integer  numfiles,
        PGSt_integer  files[])
        PGSt_integer  version[];
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_sendruntimedata(numfiles,files,version)
        integer numfiles
        integer files(*)
        integer version(*)
```

DESCRIPTION: This tool provides the user with a method for flagging specific runtime data files for subsequent post-processing retrieval.

INPUTS: numfiles-exact number of runtime logical file identifiers loaded into the array 'files'

files-array of logical file identifiers which are to be preserved for later retrieval

version-an associated array for identifying specific versions of the files identified in the preceding array of logical identifiers

OUTPUTS: None

RETURNS:

Table 6-53. PGS_SMF_SendRuntimeData Returns

Return	Description
PGS_S_SUCCESS	Success
PGSSMF_E_SENDRUNTIME_DATA	Send runtime file data error
PGSSMF_M_TRANSMIT_DISABLE	Transmission of files is disabled

EXAMPLES:

C:

```
==
/# These constants may be defined in the users include
   file(s). #/
```

```
/# Note that these logical file identifiers would have to
   appear #/
/# in the Process Control file in order for this call to
   work. #/
#define MODIS1A   10
#define MODIS2    20
#define TEMP1     50
#define TEMP2     51
#define TEMP3     52

PGSt_SMF_status   returnStatus;
PGSt_integer      numberOfFiles;
PGSt_integer      logIdArray[6];
PGSt_integer      version[6];
PGSt_integer      version_MODIS1A_1 = 1;
PGSt_integer      version_MODIS1A_2 = 2;
PGSt_integer      version_MODIS2    = 1;
PGSt_integer      version_TEMP      = 1;

logIdArray[0] = MODIS1A;  version[0] = version_MODIS1A_1;
logIdArray[1] = MODIS1A;  version[1] = version_MODIS1A_2;
logIdArray[2] = MODIS2;   version[2] = version_MODIS2;
logIdArray[3] = TEMP1;    version[3] = version_TEMP;
logIdArray[4] = TEMP2;    version[4] = version_TEMP;
logIdArray[5] = TEMP3;    version[5] = version_TEMP;
numberOfFiles = 6;

returnStatus =
PGS_SMF_SendRuntimeData(numberOfFiles,logIdArray,version);
if (returnStatus == PGS_S_SUCCESS)
{
    /# send runtime data success #/
}
```

FORTRAN:

```
C         The following constants may be defined in the users include file(s).

C         Note that the specific logical file identifiers would have to appear

C         in the process control file in order for this call to work.

          implicit none

          integer     pgs_smf_sendruntimedata
          integer     modis1a
          parameter   (modis1a = 10)
          integer     modis2
          parameter   (modis2 = 20)
```

```
                integer     temp1
                parameter   (temp1 = 50)
                integer     temp2
                parameter   (temp2 = 51)
                integer     temp3
                parameter   (temp2 = 52)

                integer     returnStatus
                integer     numberOfFiles
                integer     logIdArray(6)
                integer     version(6)
                integer     version_modis1a_1
                integer     version_modis1a_2
                integer     version_modis2
                integer     version_temp

                version_modisa_1  = 1
                version_modisa_2  = 2
                version_modis2    = 1
                version_temp      = 1

                logIdArray(1)     = modis1a
                version(1)        = version_modis1a_1

                logIdArray(2)     = modis1a
                version(2)        = version_modis1a_2

                logIdArray(3)     = modis2
                version(3)        = version_modis2

                logIdArray(4)     = temp1
                version(4)        = version_temp

                logIdArray(5)     = temp2
                version(5)        = version_temp

                logIdArray(6)     = temp3
                version(6)        = version_temp

                numberOfFiles     = 6

                return_status =
                pgs_smf_sendruntimedata(numberOfFiles,logIdArray,version)

                if (return_status .EQ. PGS_S_SUCCESS) then

C         send runtime data success
                endif
```

NOTES: Repeated calls to this tool will cause previously requested files to be superseded with the list provided during the last call.

IMPORTANT TOOLKIT NOTES

This tool does not trigger the spontaneous transmission of runtime files and e-mail notification, as it did in Toolkit 3. Rather, the requested files are saved/marked for transmission following the normal termination of the PGE process. The actual transmission procedure is performed by the termination process (See PGS_PC_TermCom() for more information on the steps required to perform this transmission).

Please refer to the documentation for PGS_PC_TermCom() for directions on how to activate/deactivate the Toolkit's transmission capability.

REQUIREMENTS: PGSTK-0630

Test Error Level

NAME: **PGS_SMF_TestErrorLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestErrorLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testerrorlevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'E'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_SMF_boolean  levelFlag;
int               *intPtr;

returnStatus = PGS_MEM_Malloc(&intPtr,sizeof(int)*10);
levelFlag = PGS_SMF_TestErrorLevel(returnStatus);
if (levelFlag
if (PGS_SMF_TestErrorLevel(returnStatus) == PGS_TRUE)
{
   /# Branch to handle error condition #/
}
else
{
   /# Some other status level returned #/
}
```

FORTRAN:

```
implicit none

INTEGER     pgs_pc_getnumberoffiles
INTEGER     returnstatus
INTEGER     numfiles
INTEGER     levelflag
PARAMETER   (ceres4 = 7090)
INTEGER     ceres4

returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
levelflag = pgs_smf_testerrorlevel(returnstatus)
IF (levelflag .EQ. PGS_TRUE) THEN

C      Branch to handle error condition
               ELSE

C      Some other status level returned
               ENDIF
```

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Fatal Level

NAME: **PGS_SMF_TestFatalLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestFatalLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testfatallevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'F'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

NOTES: NONE

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Message Level

NAME: **PGS_SMF_TestMessageLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestMessageLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testMessagelevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'M'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

NOTES: None

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

REQUIREMENTS: PGSTK-0590

Test Warning Level

NAME: **PGS_SMF_TestWarningLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestWarningLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_testwarninglevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'W'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

NOTES: None

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

REQUIREMENTS: PGSTK-0590

Test User Information Level

NAME: **PGS_SMF_TestUserInfoLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestUserInfoLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testuserinfolevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'U'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Success Level

NAME: **PGS_SMF_TestSuccessLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestSuccessLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testsuccesslevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'S'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Notice Level

NAME: **PGS_SMF_TestNoticeLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_boolean
PGS_SMF_TestNoticeLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_testnoticelevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a Boolean value indicating whether or not the returned code has level 'N'.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS: PGS_FALSE
PGS_TRUE

EXAMPLES: See example for PGS_SMF_TestErrorLevel( );

NOTES: None

REQUIREMENTS: PGSTK-0590

Test Status Level

NAME: **PGS_SMF_TestStatusLevel( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_TestStatusLevel(
        PGSt_SMF_status code);
```

FORTRAN:

```
include'PGS_SMF.f'

integer function pgs_smf_teststatuslevel(code)
        integer code
```

DESCRIPTION: Given the mnemonic status code, this tool will return a defined status level constant.

INPUTS: code-mnemonic error/status code generated by message compiler

OUTPUTS: None

RETURNS:

Table 6-54. PGS_SMF_TestStatusLevel Returns

Return	Description
PGS_SMF_MASK_LEV_S	Success level status
PGS_SMF_MASK_LEV_M	Message level status
PGS_SMF_MASK_LEV_U	User information level status
PGS_SMF_MASK_LEV_N	Notice level status
PGS_SMF_MASK_LEV_W	Warning level status
PGS_SMF_MASK_LEV_E	Error level status
PGS_SMF_MASK_LEV_F	Fatal level status
PGSSMF_E_UNDEFINED_CODE	Undefined code

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
int               *intPtr;

returnStatus = PGS_MEM_Malloc(&intPtr,sizeof(int)*10);
switch(PGS_SMF_TestStatusLevel(returnStatus))
{
   case PGS_SMF_MASK_LEV_S:
```

```
      /# This is a success level status #/
         break;

      case PGS_SMF_MASK_LEV_M:
      /# This is a message level status #/
         break;

      case PGS_SMF_MASK_LEV_U:
      /# This is a user information level status #/
         break;

      case PGS_SMF_MASK_LEV_N:
      /# This is a notice level status #/
         break;

      case PGS_SMF_MASK_LEV_W:
      /# This is a warning level status #/
         break;

      case PGS_SMF_MASK_LEV_E:
      /# This is a error level status #/
         break;

      case PGS_SMF_MASK_LEV_F:
      /# This is a fatal level status #/
         break;

      default:
      /# Undefined status level #/
         break;
   }
```

FORTRAN:

```
   implicit none

   INTEGER           pgs_pc_getnumberoffiles
   INTEGER           returnstatus
   INTEGER           numfiles
   INTEGER           levelmask
   PARAMETER         (ceres4 = 7090)
   INTEGER           ceres4

   returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
   levelmask = pgs_smf_teststatuslevel(returnstatus)
   IF (levelmask .EQ. PGS_SMF_MASK_LEV_S) THEN

C  This is a success level status
   ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_M) THEN

C  This is a message level status
   ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_U) THEN
```

```
C   This is a user information level status
                ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_N) THEN

C   This is a notice level status
                ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_W) THEN

C   This is a warning level status
                ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_E) THEN

C   This is a error level status
                ELSE IF (levelmask .EQ. PGS_SMF_MASK_LEV_F) THEN

C   This is a fatal level status
                ELSE

C   Undefined status level
                ENDIF
```

NOTES: The returned level constants are ordered by severity with PGS_SMF_MASK_LEV_S having a small integral value and PGS_SMF_MASK_LEV_F having the highest. This enables you to perform conditional tests between a particular status code and one of the provided level constants.

REQUIREMENTS: PGSTK-0590

Begin Function

NAME: **PGS_SMF_Begin( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_Begin(
        char *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_begin(funcname)
character*100 funcname
```

DESCRIPTION: A call to this tool signals to SMF that a function has started, and thus, the current message indent level should be incremented.

INPUTS:

Table 6-55. PGS_SMF_Begin Returns

Name	Description
funcname	The name of the function which calls this routine.

OUTPUTS: **NONE**

RETURNS: PGS_S_SUCCESS

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_Begin("CallingFunction");
```

FORTRAN:

```
integer pgs_smf_begin

integer returnStatus

returnStatus = pgs_smf_begin('CallingFunction')
```

NOTES: A message will be written to the status log file indicating that the specified function has started.

REQUIREMENTS: PGSTK-0580,0590,0650,0663

End Function

NAME: **PGS_SMF_End( )**

SYNOPSIS:

C:

```
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_SMF_End(
        char *funcname);
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_smf_end(funcname)
character*100 funcname
```

DESCRIPTION: A call to this tool signals to SMF that a function has completed, and thus, the current message indent level should be decremented.

INPUTS:

Table 6-56. PGS_SMF_End Returns

Name	Description
funcname	The name of the function which calls this routine.

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

returnStatus = PGS_SMF_End("CallingFunction");
```

FORTRAN:

```
implicit none

integer pgs_smf_end

integer returnStatus

returnStatus = pgs_smf_end('CallingFunction')
```

NOTES: A message will be written to the status log file indicating that the specified function has completed.

REQUIREMENTS: PGSTK-0580,0590,0650,0663

We have found that this function could not be implemented in a POSIX compliant manner across all development platforms. We note, however, that with the exception of one platform (IBM), all machines, by default, enable their own implementation-dependent floating-point exception handling features. In a general sense, these features provide the functional equivalent of the Toolkit exception handling mechanism. See "Investigation Results on the use of Signal Exception Handling for ECS Approved Computing Platforms" on the Toolkit Primer web page for more details.

NAME: **PGS_SMF_SetArithmeticTrap( )**

SYNOPSIS:

C:

```
#include <PGSSMF.h>

PGSt_SMF_status
PGS_SMF_SetArithmeticTrap(
        void (*func)(int signo));
```

FORTRAN: TBD

DESCRIPTION: This tool should be used to specify a signal handling function to be called to handle arithmetic exception events.

INPUTS: func-signal handling function

OUTPUTS: None

RETURNS:

Table 6-57. PGS_SMF_SetArithmeticTrap Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	UNIX error

EXAMPLES:

C:

```
PGSt_SMF_status returnStatus;

void SignalHandler(int signo)
{
   /# algorithm to handle SIGFPE #/
}

main( )
{
   /# initialization section #/
```

```
   returnStatus = PGS_SMF_SetArithmeticTrap(SignalHandler);
   if (returnStatus == PGS_S_SUCCESS)
   {
      /# signal trap set successfully #/
   }
   else
   {
      /# signal trap not set #/
      exitStatus = 1;
      goto EXIT;
   }
   /# main body #/
   .
   .
   .
   for (alt=5000; alt<100000; alt+500)
   {
      density[alt]=(GAS_CONST * temp[alt]) / pressure[alt];
   }
   .
   .
   .
   EXIT:
   exit( existStatus );
}     /# end main #/
```

FORTRAN: TBD

NOTES: Use NULL in place of a signal handling function to set the Toolkit default signal handling function. This handler will force an exit from the user's program, which is generally more acceptable than the system's default action (i.e., core dump).

Upon successful completion of the user's signal handling function, program control will be returned to the point where the fault occurred. As a side-effect, the default Toolkit signal handling function will be restored to safeguard against future occurrences of this event.

The user's signal handling routine must accept the integer argument for the signal number. It is not required for the user to take any action on the value; it is strictly for informational purposes only.

This tool only responds to the POSIX signal SIGFPE; all other signals need to be handled by other means.

REQUIREMENTS: PGSTK-0660

6.2.2.3 Error and Status Message File Creation Tool

Status Message File Creation

NAME: **smfcompile**

SYNOPSIS:

C: smfcompile -f textfile [-r] [-i]

smfcompile -f textfile -c [r] [i]

FORTRAN: smfcompile -f textfile -f77 [-r] [-i]

ALL: smfcompile -f textfile -all [-r] [-i]

Ada: smfcompile -f textfile -ada [-r] [-i]

DESCRIPTION: This utility generates runtime status message files and language dependent include files from user-defined status message text files.

INPUTS: textfile-status message text file (e.g., PGS_IO_100.t)

- c-create C include file
- f77-create FORTRAN include file
- all-create FORTRAN, C and Ada include files
- r-redirect the created ASCII runtime message file to the directory set in the environment variable “PGSMSG”
- i-redirect the created language-specific include file to the directory set in the environment variable “PGSINC”

OUTPUTS: Language-specific include file and ASCII runtime message file (an Ada package specification will be produced in place of an include file when the ‘-ada’ switch is used).

RETURNS: 1-error occurred

0-successful operation

EXAMPLES: smfcompile -f PGS_IO_100.t (produces PGS_IO_100.h and PGS_100)

smfcompile -f PGS_IO_100.t -c (produces PGS_IO_100.h and PGS_100)

smfcompile -f PGS_IO_100.t -f77 (produces PGS_IO_100.f and PGS_100)

smfcompile -f PGS_IO_100.t -all (produces PGS_IO_100.f,
PGS_IO_100.h, PGS_IO_100.a and PGS_100)

NOTES: The environment variable PGSMSG must be set to the local Toolkit installation directory '/../pgs/message' in order for the Toolkit to function properly. The reason for this is that Toolkit status message files will already reside in this directory upon completion of the Toolkit installation procedure; these files must be visible at runtime for the Toolkit to function properly.

If you do not specify the "-r" input parameter to the smfcompile, then make sure that the newly created ASCII runtime message file is moved to the directory set in the environment variable "PGSMSG".

REQUIREMENTS: PGSTK-0581, PGSTK-0590, PGSTK-0591, PGSTK-0600, PGSTK-0650, PGSTK-0664

6.2.3 Process Control Tools

The Process Control Tools perform the task of communicating Process Control information to the PGE. This information may consist of Production Run ID; Science Software ID; physical file names (or *Universal Reference* identifiers); input file metadata/ attributes; and PGE specific runtime parameter information. Access to this data is provided through a library API and a command-level interface, as described in detail below.

For Toolkit 5, an additional tool has been created which allows the user to query on the type of file that is of current interest. This tool, PGS_PC_GetReference, provides the user with the means to determine whether a file is of type temporary or product.

Another important change for Toolkit 5 involves the removal of most Toolkit dependency information based on environment variables. All the environment variables that define the default location for PCF information, for each PCF section (e.g., product input), have been replaced with section headers in the PCF. The means to provide this default information is still there, but the method has been changed. To reduce the number of environment variables that the user would otherwise, as in the past, be required to set.

Several new tools were added for Toolkit 4; chief among them was the product metadata retrieval tools PGS_PC_GetFileAttr and PGS_PC_GetFileByAttr. These tools provide the means to retrieve metadata that results from an inventory search; a search performed, by the Planning and Data Processing subsystem, as part of the normal processing setup prior to PGE execution. These tools should not be confused with the Metadata tools that are more specialized tools for managing the various types of metadata (See Section 6.2.1.4). These latter tools provide for the generation and association of product metadata whereas the former only provide for the retrieval of product metadata. Once the definition for metadata matures and the design for managing it in the data server becomes clearer, it may be possible to unify these tools in such a way as to provide for the greatest degree of benefit to the user.

In addition to the above, several new tools were added in Toolkit 4 to provide command, or shell, level access to most of the process control functionality delivered in Toolkit 3. This additional interface will provide for a greater degree of flexibility, when developing PGEs, by allowing the user to take advantage of standard shell level features when manipulating process control information.

However, some of these new tools have a different objective. To provide for a more seamless integration of the Toolkit with a PGE, a few command utilities have been incorporated which perform Toolkit initialization and termination procedures; these steps are necessary to support the Toolkit to its fullest extent. Since these tools are used outside of the PGE, they do not place an additional burden on the development of a PGE. The user is however encouraged to activate these tools whenever testing is performed. To provide for this eventuality, there is now a shell command that provides an integrated solution for the inclusion of these tools during PGE testing.

As newer, higher-level, tools have emerged, greater has the need become to abstract away the older, lower-level tools. To safeguard against future changes in the Toolkit API, the PGS_PC_GetPCSData and PGS_PC_PutPCSData routines were removed from the User's Guide

in Toolkit 4. This step is necessitated by the possibility of having to support a different Process Control implementation for the DAAC environment. We regret any inconvenience that this may cause.

In order for these tools to function, the actual process control information needs to be specified in a Process Control file (PCF) prior to activation of the PGE. Each Process Control file contains various subject fields to hold specific runtime information. All product/support/temporary file I/O subject fields follow a similar format; the ones that differ deal with system defined and user defined parameter information. Each subject-field entry contains a key identifier and numerous attributes that describe the particular entry.

To support testing of a PGE, the user must create entries in a PCF to account for all file inputs, all file outputs (except intermediate and temporary), and all parameter information that the particular PGE depends on. The key identifiers that name each entry, also need to be represented as logical identifiers in the PGE software. Then at runtime, the attributes for a particular entry may be retrieved by passing a specific key identifier to the appropriate PC Toolkit function. (Note that certain IO Toolkit functions access the file I/O entries when product/support/temporary file key identifiers are passed to them) For this reason, it would be prudent to create a meaningful constant identifier for each key identifier in the PCF, e.g., TEMP1=100.

This process of defining a PCF will need to be performed for every unique instance of a PGE. At runtime, these tools will access the particular PCF that is pointed to by the environment variable PGS_PC_INFO_FILE.

The measures outlined in the preceding paragraph must be performed to provide the minimal level of PGS emulation required to support the Toolkit, since many Toolkit functions rely on the Process Control mechanism for I/O and parameter information. The Process Control File 'PCF.v5,' which was delivered along with the Toolkit in directory '$PGSHOME/runtime,' contains all the necessary Toolkit dependencies, some of which may need to be customized for certain Toolkit functions. **To avoid PCF collisions between Toolkit and developer dependencies, logical identifiers in the range 10,000 to 10,999 have been reserved exclusively for Toolkit use; any other valid positive integer may be used for development purposes.**

To mediate against any potential problems caused by an improperly constructed Process Control File; an additional tool has been added which can be used by the developer to screen a PCF for syntax errors and missing Toolkit dependencies. For more information on the usage of this utility, refer to the section below for the 'pccheck' tool.

Please refer to Appendix C for guidance on the construction of Process Control Files and to examine a sample PCF. More details and examples on the usage of the 'pccheck' utility are also included in this appendix.

6.2.3.1 Process Control Command Tools

Toolkit Shell Script Command

NAME: **PGS_PC_Shell.sh**

SYNOPSIS: PGS_PC_Shell.sh [-h] <PGE file> <Init string> <PCF location> <SMF Cache Size> [-v] [-p]

C: N/A

FORTRAN: N/A

DESCRIPTION: This shell script accepts four command line arguments as input. The first argument is the PGE to run. This may be a shell script or an executable. The second argument is the Init string that contains 4 binary digits that define how the Toolkit will behave. Together, these instruct the shell about what to do in the case of using/not using shared memory or using/not using log files. The third argument is the location of the Process Control File (PCF). The forth argument is the SMF cache size. A fifth argument may be used to run this script in verbose mode. A sixth argument may be used to pass the return value of the PGE through as the return value of the script.

INPUTS: PGE file-The full path/file name of the PGE to be run

Init string-The string to be passed in with the instructions about what to do with shared memory and the log file. See NOTES section for complete description of each field in the Init string flag.

PCF location-The full path/file name of the Process Control File (PCF)

SMF Cache Size-size of SMF message cache in records

v-Run in verbose mode. Output status messages displaying settings, current file being run.

p-Make the return value of this script be the return value of the PGE if the PGE is run. If the PGE does not get run then revert to the normal method of return values for this shell.

h-Upon receiving the -h flag a short description of the usage of PGS_PC_Shell.sh will be provided to the user and the command will exit.

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM

PGS_SH_MEM_INIT
PGS_SH_PC_DELETETMP
PGS_SH_SMF_SENDRUNTIME
PGS_SH_SMF_SENDLOGFILE
PGS_SH_MEM_TERM
PGS_SH_SMF_LOGFILE
PGS_SH_PC_LOADDATA
PGS_SH_PC_ENV
PGS_SH_SMF_SHMMEM

EXAMPLES:

PGS_PC_Shell.sh -h
PGS_PC_Shell.sh /usr/PGE/somePGE 1111
/usr/PGE/data/PCF.current 50 -v
PGS_PC_Shell.sh /usr/home/PGE/runFile 1010
/home/PCFDATA/pcf.data 200
PGS_PC_Shell.sh /usr/PGEhome/runThis 0000
/home/Data/MY.pcf 150 -p

NOTES:

This shell script parses the input to ensure correctness and will report any input problems to the user.

This shell script acts as the outer most shell for the PGE.

The Init string flag consists of four (4) fields. Each field contains a single digit. The digits should be a one (1) or a zero (0). Therefore the Init String would appear as “1010” or “1111”, etc. For ease of use PGS_PC_Shell.sh will interpret any non-zero digit as a one. Therefore, 8020 would be interpreted as 1010, and 5500 would be interpreted as 1100, etc. The field descriptions are listed as follows:

FIELD 1 - 1 (or any non-zero digit) = Use shared memory if available
0 = Do not use shared memory

FIELD 2 - 1 (or any non-zero digit) = If shared memory fails continue using ASCII files
0 = If shared memory fails stop now

FIELD 3 - 1 (or any non-zero digit) = Use Log Files
0 = Do not use Log Files

FIELD 4 - 1 (or any non-zero digit) = If Log Files fail continue anyway
0 = If Log Files fail stop now

In order to enable PGS_PC_Shell.sh to delete temporary files automatically at PGE termination, one needs to call

PGS_IO_Gen_Temp_Delete within PGE or PGS_PC_TempDelCom within the PGE shell. These functions mark the temporary file for deletion (they add flag "D" to temporary files version number) in the PCF. The shell script that physically removes temporary files is PGS_PC_Term Com. This is usually the last call in the PGE shell.

REQUIREMENTS: PGSTK-1312

Toolkit Initialization Command

NAME: **PGS_PC_InitCom**

SYNOPSIS: PGS_PC_InitCom <shared-memory-flag> <log-file-flag> <num.-smf-records>

C: N/A

FORTRAN: N/A

DESCRIPTION: This program performs the initialization for the PGE.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-flag stating whether or not to use shared memory

argv[2]-flag stating whether or not to write to a log file

argv[3]-number of SMF records to store in shared memory

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_MEM_INIT
PGS_SH_SMF_LOGFILE
PGS_SH_PC_LOADDATA
PGS_SH_PC_ENV
PGS_SH_SMF_SHMMEM

EXAMPLES:
```
PGS_PC_InitCom ShmOn LogOn 50
PGS_PC_InitCom ShmOff LogOn 100
```

NOTES: This program is intended to be run from within PGS_PC_Shell.sh and is not designed to be run from the command line as a stand-alone program.

REQUIREMENTS: PGSTK-1311

Get Physical File Reference Command

NAME: **PGS_PC_GetReferenceCom**

SYNOPSIS: PGS_PC_GetReferenceCom <logical ID> <version>

DESCRIPTION: This program will retrieve the physical file reference associated with a logical ID.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the configuration parameter

argv[2]-version of the physical file reference to retrieve. A one-to-one relationship exists between all files except for product input files.

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

Get the physical file reference associated
#  with ID 12297

REFERENCE=`PGS_PC_GetReferenceCom $LogicalID $Version`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
      FILENAME=`echo $REFERENCE | cut -f1 -d" "`
      VERSIONS=`echo $REFERENCE | cut -f2 -d" "`
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
```

```
#  report an error found
fi
.
.
.
```

Another method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

#  Get the physical file reference associated
#  with ID 12297
set `PGS_PC_GetReferenceCom $LogicalID $Version`
#  The file reference and versions remaining will
#  now appear in two separate tokens.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
      FILENAME=$1
      VERSIONS=$2
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
#  report an error found
fi
.
.
.
```

A final method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297
Version=1

#  Get the physical file reference associated
#  with ID 12297
set “`PGS_PC_GetReferenceCom $LogicalID $Version`”
```

```
#  Placing double quotes around the command causes
#  the string to be placed in one token.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
      FILENAME=`echo $1 | cut -f1 -d” “`
      VERSIONS=`echo $1 | cut -f2 -d” “`
#  FILENAME now contains the file reference.
#  VERSIONS now contains the versions remaining.
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE script.

The user will be required to parse the file name and number of files remaining from the output string. This can be done using the cut command (See EXAMPLES). The file name and versions remaining will be separated by a single space.

REQUIREMENTS: PGSTK-1290

Get User Defined Configuration Parameters Command

NAME: **PGS_PC_GetConfigDataCom**

SYNOPSIS: PGS_PC_GetConfigDataCom <logical ID>

DESCRIPTION: This program will retrieve user defined configuration parameters from the PCF or shared memory at the command line.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the configuration parameter

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Get the parameter associated with ID 12297
CONFIG=`PGS_PC_GetConfigDataCom $LogicalID`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-1291

Get Number Of Files Command

NAME: **PGS_PC_GetNumberOfFilesCom**

SYNOPSIS: PGS_PC_GetNumberOfFilesCom <logical ID>

DESCRIPTION: This program will retrieve the number of product input files from the PCF or shared memory at the command line.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the product input files to be inquired

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Get the number of product files associated
#  with ID 12297
NUMFILES=`PGS_PC_GetNumberOfFilesCom $LogicalID`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-1315

Get File Attribute Command

NAME: **PGS_PC_GetFileAttrCom**

SYNOPSIS: PGS_PC_GetFileAttrCom <logical ID> <version> <format flag>

DESCRIPTION: This program will retrieve a file attribute string or location associated with a product input file from the PCF or shared memory at the command line.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the configuration parameter

argv[2]-version number of file to retrieve attribute for

argv[3]-format flag that states whether to return the attribute or the location of the file attribute. Possible values are:

PGSd_PC_ATTRIBUTE_LOCATION
PGSd_PC_ATTRIBUTE_STRING

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR
PGS_SH_PC_TRUNC

EXAMPLES: The following example is valid for the Bourne and Korn shells only.

```
#  This is within a shell script - probably within the
#  PGE script.
#  Set our format flag values. (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_PC_ATTRIBUTE_LOCATION=1}
: ${PGSd_PC_ATTRIBUTE_STRING=2}

LogicalID=12297
Version=1
FormatFlag=$PGSd_PC_ATTRIBUTE_STRING

#  Get the file attribute string associated with
#  the first file of product ID 12297
ATTR=`PGS_PC_GetFileAttrCom $LogicalID $Version $FormatFlag`
RETVAL=$?
```

```
#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  Variable ATTR now contains the attribute string
else
#  report an error found
fi
.
.
.
```

If the user wishes to use a c-shell script this is the recommended technique to use. In a c-shell script if the user fails to use this technique the script will give undefined results (see NOTES).

```
#  This is within a shell script - probably within the
#  PGE script.
#  Set our format flag values. (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
set PGSd_PC_ATTRIBUTE_LOCATION=1
set PGSd_PC_ATTRIBUTE_STRING=2

set LogicalID=12297
set Version=1
set FormatFlag=$PGSd_PC_ATTRIBUTE_STRING

#  Get the file attribute string associated with
#  the first file of product ID 12297
PGS_PC_GetFileAttrCom $LogicalID $Version $FormatFlag
>out.file
set RETVAL=$status

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  File out.file now contains the attribute string
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

If the format flag passed in is equal to PGSd_PC_ATTRIBUTE_STRING the return value is the attribute string appended as one long string. If the format flag passed in is equal to PGSd_PC_ATTRIBUTE_LOCATION the return value is the attribute location that is a full path and file name of the file containing the attribute string.

If the user wishes to use this program in a c-shell script the output of the program must be re-directed to a file and the file can then be manipulated. A long string can not be assigned to a variable in a c-shell script. Attempting to assign a long string to a variable will give undefined results in the c-shell.

REQUIREMENTS: PGSTK-1314

Get the Temporary File Reference Command

NAME: **PGS_PC_GetTempReferenceCom**

SYNOPSIS: PGS_PC_GetTempReferenceCom <logical ID> <duration of file>

DESCRIPTION: This program will retrieve a temporary file reference from the PCF. If a reference does not exist it will create one.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the temporary file reference

argv[2]-file duration

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
TEMPREFERENCE=`PGS_PC_GetTempReferenceCom $LogicalID
$Endurance`
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and existence flag
#  can be parsed.
      FILENAME=`echo $TEMPREFERENCE | cut -f1 -d" "`
      EXISTS=`echo $TEMPREFERENCE | cut -f2 -d" "`
```

```
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

Another method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE script.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
set `PGS_PC_GetTempReferenceCom $LogicalID $Endurance`
#  The file reference and existence flag will
#  now appear in two separate tokens.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
        FILENAME=$1
        EXISTS=$2
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

A final method of performing this task is as listed below. This method only works in the Korn and Bourne shells.

```
#  This is within a shell script - probably within the
#  PGE script.

#  Set our endurance values.  (This is Bourne shell format)
#  These values are set in PGS_PC_Shell.sh.
: ${PGSd_IO_Gen_NoEndurance=0}
: ${PGSd_IO_Gen_Endurance=1}

LogicalID=12297
Endurance=$PGSd_IO_Gen_NoEndurance

#  Get the temporary physical file reference associated
#  with ID 12297
set "`PGS_PC_GetTempReferenceCom $LogicalID $Endurance`"
#  Placing double quotes around the command causes
#  the string to be placed in one token.
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
#  This is how the file name and versions remaining
#  can be parsed.
       FILENAME=`echo $1 | cut -f1 -d" "`
       EXISTS=`echo $1 | cut -f2 -d" "`
#  FILENAME now contains the file reference.
#  EXISTS now contains the existence flag.
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

If a temporary file reference does not exist for the logical ID then a reference is created. The user will be able to determine if the reference existed by checking the existence flag portion of the program return (See EXAMPLES).

The user will be required to parse the file name and the existence flag from the output string. This can be done using the cut command (See EXAMPLES). The file name and the existence flag will be separated by a single space.

REQUIREMENTS: PGSTK-0531, PGSTK-0535, PGSTK-1291

Delete Temporary File Command

NAME: **PGS_PC_TempDeleteCom**

SYNOPSIS: PGS_PC_TempDeleteCom <logical ID>

DESCRIPTION: This program will flag a temporary file as deleted in the PCF or shared memory at the command line.

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-logical ID of the temporary file to be deleted

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_NODATA
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.

LogicalID=12297

#  Delete the temporary file with the logical ID 12297
PGS_PC_TempDeleteCom $LogicalID
RETVAL=$?

#  Check the return value
if [ $RETVAL -eq 0 ]
then
#  continue normal processing
else
#  report an error found
fi
.
.
.
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-0521

Get File Size Command

NAME: **PGS_PC_GetFileSizeCom**

SYNOPSIS: PGS_PC_GetFileSizeCom <logical ID>

DESCRIPTION: This program will retrieve the file size of the file associated with the input logical ID and version in the users Process Control File (PCF).

INPUTS: argc-number of command line arguments
argv[0] - logical ID (in the PCF) of the desired file
argv[1] - file version number

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_SYS_PARAM
PGS_SH_PC_TOOLERROR

EXAMPLES:

```
#  This is within a shell script - probably within the
#  PGE shell.  This example assumes there is an entry for
#  for a file in the users PCF with logical ID 101

LogicalID=101
Version=1

#  Get the physical file size associated with the user's
#  input arguments LogicalID and Version

SIZE= `PGS_PC_GetFileSizeCom $LogicalID $Version`
RETVAL=$?

#  Check the return value

if [ $RETVAL -eq 0 ]
then

#  SIZE now contains the file size.
#  continue normal processing...

            :
            :

else

#  handle error case...
            :
            :
fi
```

NOTES: This program is designed to be run from within the PGE.

REQUIREMENTS: PGSTK-1290

Toolkit Termination Command

NAME: **PGS_PC_TermCom**

SYNOPSIS: PGS_PC_TermCom <shared-memory-flag> <log-file-flag>

C: N/A

FORTRAN: N/A

DESCRIPTION: This program runs the functions necessary to clean up shared memory, send runtime files, send logfiles, update the PCF, and remove temporary files (it removes the temporary files if PGS_IO_Gen_Temp_Delete is called within PGE or PGS_PC_TempCom is called within the PGE shell).

INPUTS: argc-number of command line arguments

argv[0]-executable name (not processed but listed here anyway)

argv[1]-flag stating whether or not to use shared memory

argv[2]-flag stating whether or not to write to a log file

OUTPUTS: NONE

RETURNS: PGS_S_SUCCESS
PGS_SH_PC_DELETETEMP
PGS_SH_SMF_SENDRUNTIME
PGS_SH_SMF_SENDLOGFILE
PGS_SH_MEM_TERM

EXAMPLES:
```
PGS_PC_TermCom ShmOff LogOff
PGS_PC_TermCom ShmOn LogOff
```

NOTES: **The send file capability of PGS_PC_TermCom is SCF functionality**. This functionality will be disabled at the Release B DAACs, but will remain available to the SCF toolkit.

The PGS_PC_TermCom tool was developed two years ago to allow SCF developers to send files to other locations in the absence of a data distribution capability. This toolkit tool was not meant to replace the ECS DAAC distribution system, but to supply functionality prior to the system availability. Instrument teams can use the distribution system, by writing an ESDT for QA files. The subscription service (B.1) can then push the files to the requestor.

In the B.0 timeframe, there is no push, per se. A work-around could be to use the Version 0 Client ordering function. Or, an email message could be sent, announcing the presence of a QA file. If this message were sent to a

special account, a script could then be run to pull the QA files out of the DAAC. This is a temporary solution, prior to B.1 operation.

If a PGE Fails:. Files are marked for sending, packaged up in a Failed Production History tar file (if and only if the PGE fails), and archived on the Data Server. The SCF is then notified and can retrieve it. If the PGE succeeds, the marked files are not put into a tar file.

The SCF Functionality:

This program is designed to be run from within the PGS_PC_Shell.sh script and is not intended to be run as a stand alone program from the command line. Running this program outside the script PGS_PC_Shell.sh will give undefined results.

Since this tool now supports the transfer of status and runtime files, certain steps need to be performed by the user to ensure that this transfer operation is carried-out properly.

FILE TRANSFER SETUP

The current transfer mechanism (ftp) requires the use of a '.netrc' file, which must reside in the user's home directory on the execution host. 'ftp' accesses this file to establish a connection with the remote host. Once the connection is made, the process of performing the actual file transfer can proceed.

This file must contain information in the following format:

machine <hostname> login <username> password <userpassword>

For example:

machine adriatic login guest password anonymous

For reasons of security, the '.netrc' file should ONLY have read permission for the user, (i.e., -rw-------).

(Refer to the man pages on netrc for more information.)

PROCESS CONTROL SETUP

As part of the transfer operation, this tool also transmits a notification message to the interested parties to inform them as to the disposition of the requested runtime and status files.

As with many other Process Control tools, this tool depends on certain entries in the Process Control File. The values of these entries however are user defined according to their local environment.

Refer to the standard Process Control File to find the following entries:

10109|TransmitFlag; 1=transmit,0=disable|0
- Set to 1 to enable file/e-mail transmission.

10106|RemoteHost|<hostname>
- Host should be the same as that which appears in the '.netrc' file.

10107|RemotePath|<destination directory>
- Directory must be writeable and large enough to hold the transferred data.

10108|EmailAddresses|<list of notification addresses>
- Notification message indicates which files have been transferred and where they currently reside.

WARNING-Do not attempt to transfer files to the same host and directory that this program is running on. The original files will be deleted in accordance with the ftp protocol for sending and receiving files. That is to say that, upon determination that the destination file is the same as the source; the destination file will be removed before sending the source file.

REQUIREMENTS: PGSTK-1311

6.2.3.2 Process Control API Tools

Get a File Reference from Logical

NAME: **PGS_PC_GetReference( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetReference(
        PGSt_PC_Logical     prodID,
        PGSt_integer        *version,
        char                *referenceID)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function pgs_pc_getreference(prodid,version,referenceid)
        integer         prodid
        integer         version
        character*200   referenceid
```

DESCRIPTION: This tool may be used to obtain a physical reference (file name) from a logical identifier.

INPUTS: prodID-User defined constant identifier that internally represents the current product.

version-Version of reference to get. Remember, for standard input files there can be a many-to-one relationship.

OUTPUTS: referenceID-The actual file reference returned as a string

version-The number of versions remaining for the requested Product ID

RETURNS:

Table 6-58. PGS_PC_GetReference Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	problem while accessing PCS data

EXAMPLES:

C:

```
#define           MODIS1A 2530

PGSt_integer      version;
char              referenceID[PGSd_PC_FILE_PATH_MAX];
PGSt_SMF_status   returnStatus;

/# Get first version of the file #/
version = 1;

returnStatus =
      PGS_PC_GetReference(MODIS1A,&version,referenceID);

/# version now contains the number of versions remaining #/

if (returnStatus != PGS_S_SUCCESS)
      goto EXCEPTION;
else
{ /# perform necessary operations on file #/ }
                .
                .
                .
EXCEPTION:
      return returnStatus;
```

FORTRAN:

```
implicit none

integer           version
character*135     referenceid
integer           returnstatus
integer           pgs_pc_getreference
integer           modis1a
parameter   (modis1a = 2530)

C                 Get the first version of the file
                  version = 1

                  returnstatus = getreference(modis1a,version,referenceid)

                  if (returnstatus .ne. pgs_s_success)
                      goto 9999
                  else

C                     perform necessary operations on file
                      .
                      .
                      .
                  9999  return
```

NOTES: All reference identifier strings are guaranteed to be no greater than PGSd_PC_FILE_PATH_MAX characters in length (see PGS_PC.h).

The version returns the number of files remaining for the product group. For example, if there are eight (8) versions of a file when the user requests version one (1) the value seven (7) is returned in version. When the user requests version two (2) the value six (6) is returned in version, etc. Therefore, it is not recommended to use version as a loop counter that is also passed into PGS_PC_GetReference().

REQUIREMENTS: PGSTK-1290

Access File Reference Type from PCF

NAME: **PGS_PC_GetReferenceType( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetReferenceType(
        PGSt_PC_Logical     identifier
        PGSt_integer        *type)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function pgs_pc_getreferencetype(identifier,type)
        integer identifier
        integer type
```

DESCRIPTION: This tool may be used to ascertain the type of file reference that is associated with a logical identifier within the science software.

INPUTS: identifier-The logical identifier as defined by the user. (This value must be mapped to an actual value via the PCF.)

OUTPUTS: type-Reference types that are defined in the PGS_PC header file. Possible values are:

PGSd_PC_INPUT_FILE_NAME
PGSd_PC_OUTPUT_FILE_NAME
PGSd_PC_TEMPORARY_FILE
PGSd_PC_INTERMEDIATE_INPUT
PGSd_PC_INTERMEDIATE_OUTPUT
PGSd_PC_SUPPORT_IN_NAME
PGSd_PC_SUPPORT_OUT_NAME

RETURNS:

Table 6-59. PGS_PC_GetReferenceType Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_FILES_FOR_ID	The Product ID does not contain a physical reference.
PGSPC_E_ENVIRONMENT_ERROR	Environment variable not set
PGSPC_E_DATA_ACCESS_ERROR	Error accessing Process Control Status data

EXAMPLES:

C:

```
#define     INSTR_SCRATCH_SPACE 2001

PGSt_SMF_status returnStatus;
PGSt_PC_Logical fileIdentifier;
PGSt_integer      fileType;

fileIdentifier = INSTR_SCRATCH_SPACE;

/# getting the type attribute of a file #/

returnStatus =
      PGS_PC_GetReferenceType(fileIdentifier,&fileType);
if (returnStatus != PGS_S_SUCCESS)
{
    goto EXCEPTION;
}
else
{
    switch (fileType)
    {
    case PGSd_PC_INPUT_FILE_NAME:
    case PGSd_PC_OUTPUT_FILE_NAME:
    case PGSd_PC_SUPPORT_IN_NAME:
    case PGSd_PC_SUPPORT_OUT_NAME:
        /#
            open standard product or support file
        #/
        returnStatus = PGS_IO_Gen_Open( );
                .
                .
                .
        break;

    case PGSd_PC_INTERMEDIATE_INPUT:
    case PGSd_PC_INTERMEDIATE_OUTPUT:
    case PGSd_PC_TEMPORARY_FILE:
        /#
            open temporary or intermediate file
        #/
            returnStatus = PGS_IO_Gen_Temp_Open( );
                .
                .
                .
            break;
    default:
```

```
                        /#
                            invalid type returned only in the event that
                            call to *GetReferenceType was not successful
                        #/

                    } /# end switch (fileType) #/
                }
                            .
                            .
                            .
                EXCEPTION:
                        return returnStatus;
```

FORTRAN:

```
                implicit none

                INTEGER INSTR_SCRATCH_SPACE
                PARAMETER (INSTR_SCRATCH_SPACE = 2001)

                integer returnstatus
                integer fileidentifier
                integer filetype
                integer pgs_pc_getreferencetype

                fileidentifier = INSTR_SCRATCH_SPACE

C           getting the type attribute of a file

                returnstatus =
                      pgs_pc_getreferencetype(fileidentifier,filetype)
                if (returnstatus .ne. pgs_s_success) then
                    goto 9999
                else if (
                        (filetype .eq. PGSd_PC_INPUT_FILE_NAME) .or.
                        (filetype .eq. PGSd_PC_OUTPUT_FILE_NAME) .or.
                        (filetype .eq. PGSd_PC_SUPPORT_IN_NAME) .or.
                        (filetype .eq. PGSd_PC_SUPPORT_OUT_NAME)
                        ) then

C           open standard product or support file

                        returnstatus = PGS_IO_Gen_OpenF(...);
                                .
                                .
                                .
                else if (
                        (filetype .eq. PGSd_PC_INTERMEDIATE_INPUT) .or.
                        (filetype .eq. PGSd_PC_INTERMEDIATE_OUTPUT) .or.
                        (filetype .eq. PGSd_PC_TEMPORARY_FILE)
                        ) then
```

```
C            open temporary or intermediate file

                        returnstatus = PGS_IO_Gen_Temp_OpenF(...);
                                .
                                .
                                .
                    else

C            invalid type returned only in the event that
C            call to *GetReferenceType was not successful

                    endif

9999    return
```

NOTES: This tool will return the reference type (mode) for files that have references in a Process Control File (PCF). This tool will not identify runtime parameters as such.

In order for this tool to function properly, a valid Process Control File will need to be created first. Please refer to Appendix C (User's Guide) for instructions on how to create and validate such a file.

REQUIREMENTS: PGSTK-1290.

Generate a Unique ID

NAME: **PGS_PC_GenUniqueID( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GenUniqueID(
        PGSt_PC_Logical     prodID,
        char                *uniqueID)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_genuniqueid(prodid,uniqueid)
        integer         prodid
        character*200   uniqueid
```

DESCRIPTION: This tool may be used to generate a unique product identifier. This identifier may be attached to file metadata to facilitate tracking of production output. The identifier may include Production Run ID, the Science Software Program ID, and the actual Product ID.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

OUTPUTS: uniqueID-The unique ID generated by this function. This ID will be returned as a string. The ID is guaranteed to be no greater than PGSd_PC_LABEL_SIZE_MAX in length (see PGS_PC.h).

RETURNS:

Table 6-60. PGS_PC_GenUniqueID Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLES:

C:

```
#define           CERES3A 300

PGSt_SMF_status   returnStatus;
char              uniqueID[PGSd_PC_LABEL_SIZE_MAX];

returnStatus = PGS_PC_GenUniqueID(CERES3A,uniqueID);
```

```
                if (returnStatus != PGS_S_SUCCESS)
                        goto EXCEPTION;
                else
                {
/# attach uniqueID into file metadata field #/
                }
                                .
                                .
                                .
                EXCEPTION:
                         return returnStatus;
```

FORTRAN:

```
                implicit none

                integer           returnstatus
                character*200     uniqueid
                integer           pgs_pc_genuniqueid
                integer           ceres3a
                parameter         (ceres3a = 300)

                returnstatus = pgs_pc_genuniqueid(ceres3a,uniqueid)

                if (returnstatus .ne. pgs_s_success) then
                        goto 9999
                else
C         attach uniqueid into file metadata field
                endif
                        .
                        .
                        .

                return
```

NOTES: If more than one product is being generated from the same PGE, then the appropriate product identifier must be used as input to this function when called from within the science software. Upon entry into this function all input values will be checked to determine that legal values were passed in. If any value is illegal, the function will return the proper error value to the calling function. All unique identifier strings are guaranteed to be no greater than PGSd_PC_LABEL_SIZE_MAX characters in length (see PGS_PC.h).

REQUIREMENTS: PGSTK-1280.

Get User Defined Configuration Values

NAME: **PGS_PC_GetConfigData( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetConfigData(
        PGSt_PC_Logical     configParamID,
        char                *configParamVal)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function pgs_pc_getconfigdata(configparamid,
*                                     configparamval)
        integer         configparamid
        character*200   configparamval
```

DESCRIPTION: This tool may be used to import run-time configuration parameters into the PGE.

INPUTS: configParamID-User defined constant that internally represents a configuration parameter.

OUTPUTS: configParamVal-A string representation of the configuration parameter value. No interpretation of this value will be done in the Toolkit; the value returned will be left to the application programmer.

RETURNS:

Table 6-61. PGS_PC_GetConfigData Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_CONFIG_FOR_ID	no configuration data for product id
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLES:

C:

```
#define           MODIS1A_CONFIG1 2990

char              configParamVal[PGSd_PC_VALUE_LENGTH_MAX];
PGSt_SMF_status   returnStatus;
long              config1;
```

```
returnStatus =
      PGS_PC_GetConfigData(MODIS1A_CONFIG1,configParamVal);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{
   /# MODIS1A_CONFIG1 is integral parameter #/
          config1 = atoi(configParamVal);

          if (config1 > 0)
          {
                  /# activate sub-process A #/
          }
          else
          {
                  /# activate sub-process B #/
          }
}
                  .
                  .
                  .

EXCEPTION:
        return returnStatus;
```

FORTRAN:

```
implicit none

character*200     configparamval
integer           returnstatus
integer           pgs_pc_getconfigdata
integer           config1
integer           modis1a_config1
parameter         (modis1a_config1 = 2990)

returnstatus =
      pgs_pc_getconfigdata(modis1a_config1,configparamval)

if (returnstatus .ne. success) then
        goto 9999
else

C
C                 modis1a_config1 is integral parameter
C                 assuming you have a function to convert character
C                 data to integer data - called.....strtoint.
C                 strtoint(configparamval,config1)
```

```
                        if (config1 .gt. 0) then
C                               activate sub-process A
                        else
C                               activate sub-process B
                        .
                        .
                        .
                        endif

                endif

                return
```

NOTES: All configuration parameter value strings are guaranteed to be less than PGSd_PC_VALUE_LENGTH_MAX characters in length (see PGS_PC.h). There will be a shell script command version of this routine to retrieve configuration information from the script.

REQUIREMENTS: PGSTK-1290.

Get Number of Files Associated with a Product

NAME: **PGS_PC_GetNumberOfFiles( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetNumberOfFiles(
        PGSt_PC_Logical     prodID,
        PGSt_integer        *numFiles)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getnumberoffiles(prodid,numfiles)
        integer prodid,
        integer numfiles)
```

DESCRIPTION: This tool may be used to determine the number of files that are associated with a particular Product ID. A many-to-one relationship may exist with Product Input, Product Output Support Input and Support Output files. This function will give the user a way to determine how many files exist for a product ID.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

OUTPUTS: numberOfFiles-Total number of files for a particular product ID.

RETURNS:

Table 6-62. PGS_PC_GetNumberOfFiles Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_FILES_FOR_ID	incorrect number of configuration parameters
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLE:

C:

```
#define     CERES4 7090

PGSt_integer      numFiles;
PGSt_integer      version;
```

```
PGSt_SMF_status   returnStatus;
int               loopCounter;
char              ceresFiles[10][PGSd_PC_FILE_PATH_MAX];

returnStatus = PGS_PC_GetNumberOfFiles(CERES4,&numFiles);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{

/# loop and get file names #/

      for (loopCounter = 0; loopCounter < numFiles;
            loopCounter++)
      {

/# specify which file to get #/

version = loopCounter + 1;

/# save references for future use #/

            returnStatus =
                  PGS_PC_GetReference(CERES4,&version,
                        ceresFiles[loopCounter]);
      }
}
            .
            .
            .
EXCEPTION:
       return returnStatus;
```

FORTRAN:

```
implicit none

integer           numfiles
integer           version
integer           returnstatus
integer           loopcounter
character*355     referenceid
character*355     ceresfiles(10)
integer           pgs_pc_getnumberoffiles
integer           pgs_pc_getreference
integer           ceres4
parameter         (ceres4 = 7090)

returnstatus = pgs_pc_getnumberoffiles(ceres4,numfiles)
```

```
      if (returnstatus .ne. pgs_s_success)
            goto 9999
      else
            do 100 loopcounter = 1,numfiles
                  version = loopcounter
                  returnstatus = pgs_pc_getreference(ceres4,
      *                                    version,
      *                                    ceresfiles(loopcounter))
100         continue
            .
            .
            .
9999  return
```

NOTES: This function will allow a one-to-many relationship to exist between logical and physical file name. The file version number is returned in reverse order. For example, if there are eight (8) versions of a Product ID and the user requests the first one, the value eight (8) would be returned in numFiles.

REQUIREMENTS: PGSTK-1290

Get the Attribute of the File Associated with the Particular Product ID and Version

NAME: **PGS_PC_GetFileAttr( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetFileAttr(
        PGSt_PC_Logical     prodID,
        PGSt_integer        version
        PGSt_integer        formatFlag,
        PGSt_integer        maxSize,
        char                *fileAttribute)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function pgs_pc_getfileattr(prodid,version,formatflag,fileAttribute)
        integer prodid
        integer version
        integer formatflag
        integer maxSize
        character*(*)   fileAttribute
```

DESCRIPTION: This tool may be used to retrieve an attribute associated with a particular product ID and version number. The data placed in the attribute will be defined and interpreted by the user. The SDP Toolkit has no dependency on the attribute.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

version-The particular version of the Product ID that the attribute is being requested from. With files there may be a many-to-one relationship.

formatFlag-Flag indicating method of attribute return. Possible values are:

PGSd_PC_ATTRIBUTE_LOCATION
PGSd_PC_ATTRIBUTE_STRING

maxSize-Amount of space allocated for attribute if formatFlag is PGSd_PC_ATTRIBUTE_STRING.

OUTPUTS: fileAttribute-The actual file attribute

If formatFlag is PGSd_PC_ATTRIBUTE_LOCATION then fileAttribute will return the file containing the attribute.

If formatFlag is PGSd_PC_ATTRIBUTE_STRING then fileAttribute will return the attribute as a string.

RETURNS:

Table 6-63. PGS_PC_GetFileAttr Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	no reference found matching product id and version number
PGSPC_W_ATTR_TRUNCATED	not enough space passed in for attribute
PGSPC_W_NO_ATTR_FOR_ID	a physical reference was found but no attribute exists for that reference
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data
PGSPC_E_INVALID_MODE	invalid format flag value passed in

EXAMPLE:

C:

```
#define           MODIS1A 4220

PGSt_integer      version;
PGSt_integer      maxSize;
PGSt_SMF_status   returnStatus;
char              fileAttribute[PGSd_PC_FILE_PATH_MAX];

version = 1;
maxSize = 0;

/# get the attribute file name of the first MODIS1A file #/

returnStatus = PGS_PC_GetFileAttr(MODIS1A,version,
      PGSd_PC_ATTRIBUTE_LOCATION,maxSize,fileAttribute);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{

/# open attribute file and search attribute for particular
   data #/

}
```

.

```
                        .
                        .
                EXCEPTION:
                        return returnStatus;
```

FORTRAN:

```
                implicit none

                integer           version
                integer           returnstatus
                integer           maxsize
                character*355     fileattribute
                integer           pgs_pc_getfileattr
                integer           modis1a
                parameter         (modis1a = 4220)

                version = 1
                maxsize = 355

C         get the attribute file name of the first modis1a file

                returnstatus = pgs_pc_getfileattr(modis1a,version,
                      PGSd_PC_ATTRIBUTE_LOCATION,maxsize,fileattribute)

                if (returnstatus .ne. pgs_s_success) then
                        goto 9999

                else

C                       open attribute file and search attribute for
C                       particular data

                endif
                        .
                        .
                        .
                return
```

NOTES: Allocating enough space for the attribute variable will be the responsibility of the application programmer. This function will write the attribute into fileAttribute for maxSize bytes or the end of the attribute, which ever comes first.

REQUIREMENTS: PGSTK-1290, PGSTK-1310

Get the Version Number of the Particular File Matching the Attribute

NAME: **PGS_PC_GetFileByAttr( )**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetFileByAttr(
        PGSt_PC_Logical     prodID,
        PGSt_integer        (*searchFunc)(char *attr),
        PGSt_integer        maxSize,
        PGSt_integer        *version)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function
pgs_pc_getfilebyattr(prodid,searchfunc,
*                    maxsize,version)
        integer prodid
        integer searchfunc
        integer maxSize
        integer version
```

DESCRIPTION: This tool may be used to retrieve the version number associated with a file with a particular attribute.

INPUTS: prodID-The logical identifier as defined by the user. The user's definitions will be mapped into actual identifiers during the Integration & Test procedure.

searchFunc-A user defined function that performs the search on the attribute. This function must be passed in as a type PGSt_integer function. It should return type PGSd_PC_MATCH upon a successful attribute match or PGSd_PC_NO_MATCH upon an unsuccessful attribute match.

maxSize-Maximum amount of space to place into attribute.

OUTPUTS: version-The version number of the file with the successful attribute match

RETURNS:

Table 6-64. PGS_PC_GetFileByAttr Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_ATTR_MATCH	did not find a match with the specified product ID
PGSPC_W_NO_ATTR_FOR_ID	the product ID contains no attribute
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data

EXAMPLE:

C:

```
#define MODIS1A   5775

PGSt_integer      searchfunc_(char *attr);   /# function
                                                prototype #/

/# The function passed into PGS_PC_GetFileByAttr() MUST be
   called #/
/# searchfunc_#/

PGSt_integer      maxSize;
PGSt_integer      version;
PGSt_SMF_status   returnStatus;
char              referenceID[PGSd_PC_FILE_PATH_MAX];

maxSize = 300;

returnStatus = PGS_PC_GetFileByAttr(MODIS1A,searchfunc_,
                                    maxSize,&version);

if (returnStatus != PGS_S_SUCCESS)
        goto EXCEPTION;
else
{

/# get file reference #/

      returnStatus =
            PGS_PC_GetReference(MODIS1A,version,referencID);

}
                .
                .
                .
EXCEPTION:
        return returnStatus;
```

FORTRAN:

```
      implicit none

      integer     version
      integer     searchfunc

C     The function passed into pgs_pc_getfilebyattr() MUST be called searchfunc

      integer           maxsize
      integer           returnstatus
      integer           pgs_pc_getfilebyattr
      integer           pgs_pc_getreference
      character*355     referenceid
      integer           modis1a
      parameter         (modis1a = 5775)

      maxsize = 300

      returnstatus = pgs_pc_getfilebyattribute(modis1a,
     *                searchfunc,maxsize,version)

      if (returnstatus .ne. pgs_s_success) then
              goto 9999
      else

C
C     get file reference
C
              returnstatus = pgs_pc_getreference(modis1a,version,
     *                                referenceid)
      endif
              .
              .
              .
      return
```

NOTES: The attribute checking is left to the application programmer. The attribute for comparison must be passed into searchFunc by means of a global variable. The attribute to be compared against will be passed into searchFunc by the function PGS_PC_GetFileByAttr( ). The function searchFunc must have declared a variable large enough to handle the incoming attribute. The attribute will be read until maxSize bytes or end of file, which ever come first.

REQUIREMENTS: PGSTK-1290

Check Process Control Information File (PCF)

NAME: **pccheck.sh**

SYNOPSIS: pccheck.sh [-h] <-i user-PCF> [-o numbered-PCF] [-c standard PCF] [-s]

C: N/A

FORTRAN N/A

DESCRIPTION: The purpose of this tool is to assist the developer in setting up a Process Control File (PCF). This utility will help to point out simple syntax and content errors that might lead to more serious runtime errors, if left uncorrected. This tool will not, however, detect errors in logic, nor will it correct PCF files.

INPUTS: -i <PCF>-The -i flag will be followed by the Process Control Information File. This flag is mandatory.

- o <outfile>-The -o flag will be followed by a file name that will be output by this command. The name of output file must be a file that does not already exist. This flag is optional.
- h-Upon receiving the -h flag a short description of the usage of pccheck.sh will be provided to the user and the command will exit.
- c-The -c option will cause a compare to be run against a specified template file. The compare will only compare the reserved Product ID's.
- s-The -s flag will cause all output except for the output from the -c flag to be suppressed.

OUTPUTS: NONE

RETURNS: 0 - Normal completion
1 - Error condition

EXAMPLE:

```
pccheck.sh -i $PGSHOME/runtime/pcf.fil -o out.fil
pccheck.sh -o out.fil -i $PGSHOME/runtime/pcf.fil
pccheck.sh -i $PGSHOME/runtime/pcf.fil -o out.fil -c
      $PGSRUN/PC/PCF.v3
pccheck.sh -i $PGSHOME/runtime/pcf.fil -c $PGSRUN/PC/PCF.v3
      -s
pccheck.sh -i in.fil
pccheck.sh -h
```

NOTES: This shell script accepts an input file (PCF) and an optional output file. The output file will be an exact copy of the input file except that line numbers are inserted into the file. This output file is provided as a convenience to the user when analyzing the generated report, which sometimes references line locations in the original PCF. This utility is also capable of comparing against a "standardized" PCF file to detect changes that have been made to the SDP Toolkit specific records (those with reserved logical identifiers in the 10K-11K range); the optional suppression flag prevents all output, other than the comparison results, from being reported.

REQUIREMENTS: PGSTK-1313

Get Universal Reference from Logical

NAME: **PGS_PC_GetUniversalRef()**

SYNOPSIS:

C:

```
#include <PGS_PC.h>

PGSt_SMF_status
PGS_PC_GetUniversalRef(
        PGSt_PC_Logical     prodID,
        PGSt_integer*       version,
        char                *universalRef)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_PC.f'
include 'PGS_PC_9.f'

integer function
pgs_pc_getuniversalref(prodid,version,universalref)
      I nteger  prodid
integer version
character*150 universalref
```

DESCRIPTION: This tool may be used to obtain a universal reference from a logical identifier.

INPUTS: prodID-User defined constant identifier that internally represents the current product.

version-Version of reference to get. Remember, for Product Input files and Product Output files there can be a many-to-one relationship.

OUTPUTS: universalRef-The actual universal reference returned as a string.

RETURNS:

Table 6-65. PGS_PC_GetReference Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	problem while accessing PCS data
PGSPC_W_NO_UREF_DATA	the product id and version contains no universal reference data

EXAMPLES:

C:

```
#define     MODIS1A 2530

PGSt_integer      version;
char  universalRef[PGSd_PC_UREF_LENGTH_MAX];
PGSt_SMF_status   returnStatus;

/# Get first version of the file #/
version = 1;

returnStatus =
PGS_PC_GetUniversalRef(MODIS1A,version,universalRef);

if (returnStatus != PGS_S_SUCCESS)
      goto EXCEPTION;
else
{ /# perform necessary operations on file #/ }
      .
      .
      .
EXCEPTION:
      return returnStatus;
```

FORTRAN:

```
      IMPLICIT NONE

      integer     version
      character*150     universalRef
      integer     returnstatus
      integer     pgs_pc_getuniversalref
      integer     modis1a
      parameter   (modis1a = 2530)

C     Get the first version of the file
      version = 1

      returnstatus =
      pgs_pc_getuniversalref(modis1a,version,referenceid)
      if (returnstatus .ne. pgs_s_success)
            goto 9999
      else

C     perform necessary operations on file
            .
            .
```

```
      .
9999  return
```

NOTES: All reference identifier strings are guaranteed to be no greater than PGSd_PC_UREF_LENGTH_MAX characters in length (see PGS_PC.h).

The version returns the number of files remaining for the product group. For example, if there are eight (8) versions of a file, when the user requests version one (1) the value seven (7) is returned in version. When the user requests version two (2) the value six (6) is returned in version, etc. Therefore, it is not recommended to use version as a loop counter that is also into PGS_PC_GetReference().

REQUIREMENTS: PGSTK-1290

Get Size of a File

NAME: **PGS_PC_GetFileSize()**

SYNOPSIS:

C:

```
#include <PGS_PC.h>
#include <PGS_SMF.h>

PGSt_SMF_status
PGS_PC_GetFileSize(
        PGSt_PC_Logical     prodID,
        PGSt_integer        version,
        PGSt_integer*       filesize)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_PC.f'
include'PGS_PC_9.f'

integer function pgs_pc_getfilesize(prodid,version,filesize)
        integer prodid,
        integer version,
        integer filesize)
```

DESCRIPTION: This tool may be used to obtain the size of a file from a logical identifier.

INPUTS: prodID-The logical identifier as defined by the user.
version - Version of reference to get.

OUTPUTS: filesize - The size of a file.

RETURNS:

Table 6-66. PGS_PC_GetFileSize Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSPC_W_NO_REFERENCE_FOUND	link number does not have the data that mode is requesting
PGSPC_E_DATA_ACCESS_ERROR	error accessing PCS data
PGS_E_UNIX	Unix system error
PGS_E_TOOLKIT	an unexpected error occurred

EXAMPLE:

C:

```
#define     PROD_ID 10501

PGSt_integer      version;
PGSt_integer      filesize;
PGSt_SMF_status   returnStatus;

/# Get first version of the file #/
version = 1;

returnStatus =
PGS_PC_GetFileSize(PROD_ID,version,&filesize);

/# version now contains the number of versions remaining #/

if (returnStatus != PGS_S_SUCCESS
goto EXCEPTION;

else

{ /# perform necessary operations on file #/ }

      .

      .

      .

EXCEPTION:

      return returnStatus;
```

FORTRAN:

NOTES: In order for this tool to function properly, a valid Process Control file will need to be created first. Please refer to Appendix C (User's Guide) for instructions on how to create such a file.

REQUIREMENTS: PGSTK-1290

6.2.4 Shared Memory Management Tools

The tools described in this section provide for a limited use of shared memory amongst executables within a PGE. These tools allow for the creation of a single user memory segment within a PGE, and for the subsequent attachment and detachment of that memory segment to another executable within the same PGE. Due to the way in which shared memory is accessed, the APIs for the C and FORTRAN programming languages are necessarily different. C users may directly manipulate the shared memory area but FORTRAN users are limited to copying to and from the shared memory area via intermediary Toolkit functions. **Note that the operation of these tools is contingent on the assumption that the user will make proper use of the initialization and termination commands that have been provided with this release of the Toolkit (please note that the Memory Management initialization and termination routines supplied with Toolkit 3 have been subsumed by corresponding Process Control commands that MUST be invoked before and after the execution of the PGE respectively). The shell utility PGS_PC_Shell.sh already activates the initialization and termination commands, so user activation of these commands should not be performed if the shell utility is used.**

Create Shared Memory Segment

NAME: **PGS_MEM_ShmCreate( )**

SYNOPSIS:

C:

```
#include <PGS_MEM1.h>

PGSt_SMF_status
PGS_MEM_ShmCreate(
        PGSt_uinteger size);
```

FORTRAN:

```
integer function pgs_mem_shmcreate(size)
integer size
```

DESCRIPTION: This tool may be used to create a shared memory segment. This tool should only be called once in a given processing script (PGE).

INPUTS size-size of the shared memory segment in bytes

OUTPUTS: None

RETURNS:

Table 6-67. PGS_MEM_ShmCreate Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment Variable “PGSMEM_SHM_SYSKEY” is not set
PGSMEM_E_SHM_MAXSIZE	Maximum system-imposed shared memory exceeded
PGSMEM_E_SHM_MULTICREATE	More than one shared-memory is created for a given PGE

EXAMPLES:

C:

```
typedef struct
{
   int  id;
   char msg[100];
}TestStruct;

TestStruct  *shmPtr;
PGSt_SMF_status  returnStatus;

returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct);
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
```

```
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,”Writing data into shared memory”);
   }
}
```

FORTRAN:

```
integer       pgs_mem_shmcreate

integer       returnstatus
integer       shm_size
character*100 test_string
shm_size = 100
test_string = “Writing data into shared memory”

returnstatus = pgs_mem_shmcreate(shm_size)
if (returnstatus .eq. pgs_s_success) then
    returnstatus = pgs_mem_shmwrite(test_string, shm_size)
endif

!  the contents of test_string have been written to shared
!  memory which can be accesses by another process in the
!  PGE
```

NOTES: This shared memory scheme is not A POSIX implementation and will therefore be subjected to change when the POSIX.4 implementation is available. System limitations will define the amount of memory that can be allocated as a shared-memory segment. Only one memory segment may be created per PGE; it may however be attached/detached as many times as are required.

REQUIREMENTS: PGSTK-1241

Attach Shared Memory Segment

NAME: **PGS_MEM_ShmAttach( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_ShmAttach(
        void **shm);
```

FORTRAN: None

DESCRIPTION: This tool may be used by an executable to attach to an existing shared memory segment. PGS_MEM_ShmCreate( ) should already be called, either within the same executable or from an earlier executable within the PGE. If the shared memory segment has been detached by calling PGS_MEM_ShmDetach( ), then you may re-attach the segment to your process-space again.

INPUTS: shm-pointer referencing the shared memory segment

OUTPUTS: shm-pointer referencing the shared memory segment

RETURNS:

Table 6-68. PGS_MEM_ShmAttach Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	Shared-memory has not been attached to the process
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process

EXAMPLES:

```
typedef struct
{
   int      id;
   char     msg[100];
}TestStruct;

PGSt_SMF_status  returnStatus;
TestStruct  *shmPtr;
```

PROCESS A:

```
returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct));
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,”From Process A”);
   }
}
```

PROCESS B:

```
returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
if (returnStatus == PGS_S_SUCCESS)
{
   if ((shmPtr->id = 123) && (strcmp(shmPtr->msg,”From
                              Process A”) == 0))
   {
      printf(“Reading data from Process A successful”);
   }
}
```

NOTES: Before using this function, PGS_MEM_ShmCreate( ) should have already be called, either within the same executable or from an earlier executable within the PGE. If the shared memory segment has been detached by calling PGS_MEM_ShmDetach( ), then you may re-attach the segment to your process-space again.

This tool lets the system select the memory location for your shared memory, thereby allowing the system to make the best possible use of its memory resources.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

REQUIREMENTS: PGSTK-1241

Detach Shared Memory Segment

NAME: **PGS_MEM_ShmDetach( )**

SYNOPSIS:

C:

```
#include <PGS_MEM1.h>

PGSt_SMF_status
PGS_MEM_ShmDetach(
        void);
```

FORTRAN: None

DESCRIPTION: This tool may be used to detach a shared memory segment from a process that it has been attached to.

INPUTS: None

OUTPUTS: None

RETURNS:

Table 6-69. PGS_MEM_ShmDetach Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_NOTATTACH	Shared-memory has not been attached to the process

EXAMPLES:

```
typedef struct
{
   int      id;
   char     msg[100];
}TestStruct;

PGSt_SMF_status  returnStatus;
TestStruct  *shmPtr;

returnStatus = PGS_MEM_ShmCreate(sizeof(TestStruct));
if (returnStatus == PGS_S_SUCCESS)
{
   returnStatus = PGS_MEM_ShmAttach((void **)&shmPtr);
   if (returnStatus == PGS_S_SUCCESS)
   {
      shmPtr->id = 123;
      strcpy(shmPtr->msg,"Writing data into shared memory");
```

```
        PGS_MEM_ShmDetach();
    }
}
```

NOTES: Note that this tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available. This function will only detach the shared memory segment from the process. The shared memory segment will not be removed from the system by calling this tool; therefore one can re-attach it again.

REQUIREMENTS: PGSTK-1241

Read from Shared Memory Segment

NAME: **PGS_MEM_ShmRead( )**

SYNOPSIS:

C: None

FORTRAN:
```
include 'PGS_SMF.f
include 'PGS_MEM_9.f'

integer function pgs_mem_shmread(mem_ptr, size)
      integer      size
      character    mem_ptr(size)
```

DESCRIPTION: This function copies the contents of shared memory into a user allocated (may be dynamically or statically allocated) memory area. This function is meant to be used by FORTRAN (77/90) users who cannot take advantage of the C shared memory tools PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

INPUTS:

Table 6-70. PGS_MEM_ShmRead Inputs

Name	Description
size	size (in bytes) of mem_ptr (see below)

OUTPUTS:

Table 6-71. PGS_MEM_ShmRead Outputs

Name	Description
mem_ptr	array or structure to which the contents of the shared memory area will be written

RETURNS:

Table 6-72. PGS_MEM_ShmRead Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	User defined shared-memory has not been created
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process
PGSMEM_E_SHM_NOTATTACH	Failed to attach shared memory to this process shared-memory

EXAMPLES:

FORTRAN:

```
integer    pgs_mem_shmread
integer    size

character  shm_buffer(1000)

integer    returnstatus

returnstatus = pgs_mem_shmread(shm_buffer, size)

if (returnstatus .ne. pgs_s_success) goto 999

!  the contents of shared memory (which may contain data
!  from a previous process) have been copied to shm_buffer

999    continue   ! process error conditions
```

NOTES:

This tool is meant to be used by FORTRAN (77/90) users ONLY. C users should use the functions PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

The tool PGS_MEM_ShmCreate() MUST be called before PGS_MEM_ShmRead() is invoked.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

The user passes in a pointer to a user defined memory area (an area of memory which has been either statically or dynamically allocated by the user) and the size of that area. This function will retrieve the pointer to the shared memory area and copy the contents of the shared memory into the users memory area. This function will then detach the shared memory from the current process. Before exiting from the PGE, the system will make sure that the attached shared memory segment will be removed from the system.

REQUIREMENTS: PGSTK-1241

Write to Share Memory Segment

NAME: **PGS_MEM_ShmWrite( )**

SYNOPSIS:

C: None

FORTRAN:
include 'PGS_SMF.f'
include 'PGS_MEM_9.f'

```
integer function pgs_mem_shmwrite(mem_ptr, size)
integer         size
character       mem_ptr(size)
```

DESCRIPTION: This function copies the contents of a user allocated (may be dynamically or statically allocated) memory area into shared memory. This function is meant to be used by FORTRAN (77/90) users who cannot take advantage of the C shared memory tool PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

INPUTS:

Table 6-73. PGS_MEM_ShmWrite Inputs

Name	Description
mem_ptr	array or structure the contents of which will be written to the shared memory area
size	size (in bytes) of mem_ptr (see above)

OUTPUTS: NONE

RETURNS:

Table 6-74. PGS_MEM_ShmWrite Returns

Return	Description
PGS_S_SUCCESS	Success
PGS_E_UNIX	
PGSMEM_E_SHM_ENV	Environment variable PGSMEM_SHM_SYSKEY is not set
PGSMEM_E_SHM_NOTCREATE	User defined shared-memory has not been created
PGSMEM_E_SHM_MULTIATTACH	Multiply attached shared-memory in a process
PGSMEM_E_SHM_NOTATTACH	Failed to attach shared memory to this process shared-memory

EXAMPLES:

FORTRAN:

```
integer    pgs_mem_shmwrite

integer    size
integer    returnstatus

character  shm_buffer(1000)

!  fill shm_buffer with interesting data

returnstatus = pgs_mem_shmwrite(shm_buffer, size)

if (returnstatus .ne. pgs_s_success)  goto 999

!  the contents of shm_buffer have been written to the
!  shared memory area which can be accessed by a subsequent
!  process

999    continue   ! process error conditions
```

NOTES: This tool is meant to be used by FORTRAN (77/90) users ONLY. C users should use the functions PGS_MEM_ShmAttach() and PGS_MEM_ShmDetach().

The tool PGS_MEM_ShmCreate() MUST be called before PGS_MEM_ShmWrite() is invoked.

This tool is not part of POSIX and is subjected to change when the POSIX.4 implementation becomes available.

The user passes in a pointer to a user defined memory area (an area of memory which has been either statically or dynamically allocated by the user) and the size of that area. This function will retrieve the pointer to the shared memory area and write the contents of the users memory area to the shared memory area OVERWRITING whatever was previously in the shared memory area. This function will then detach the shared memory from the current process. Before exiting from the PGE, the system will make sure that the attached shared memory segment will be removed from the system.

REQUIREMENTS: PGSTK-1241

6.2.5 Bit Manipulation Tools

It is assumed that bit-manipulation functionality will be provided inherently by the language for 'C' and Fortran90 and that users of Fortran77 will use compilers that conform to MIL STD 1753 to obtain these capabilities.

6.2.6 Spacecraft Ephemeris and Attitude Data Access Tools

This tool group contains tools and associated software that provides access to the spacecraft ephemeris and attitude at a given time. Currently the EOS_AM, EOS_PM, EOS_AURA and TRMM platforms are supported. In this release of the Toolkit, orbit and attitude data for testing is supplied by the ECS Spacecraft Orbit and Attitude Simulator. Both binary and HDF formats for orbit and attitude data is supported. The binary orbit and attitude data files can be produced on a platform of "big" or "little" endian type. Toolkit will swap the eph and att data after reading binary files if data files endianness do not agree with the platform's endianness.

6.2.6.1 Orbit and Attitude Simulator

The ECS Spacecraft Orbit and Attitude Simulator is based on Upper Atmosphere Research Satellite (UARS) FORTRAN code. It has been completely rewritten in C and revised for EOS.

6.2.6.1.1 Brief Description

The spacecraft orbit simulator *orbsim* will create files (binary and HDF) of simulated spacecraft orbit and attitude data necessary to test the SDP Toolkit spacecraft ephemeris and attitude data access tool (PGS_EPH_EphemAttit( )) in the SCF environment. For platforms such as DEC alpha and PC the binary files will be substituted automatically with the big-endian binary type data files contained in the testdriver tar file upon running the test shell script runTest. This is for testing cross endiannes of data files and test platform. Users may alternatively create their own data files, either on a big-endian or little-endian machines, but MUST follow the ECS ephemeris and attitude file formats.

WARNING: this simulator uses a relatively simple algorithm and is meant to produce data for software testing ONLY. This data should not be used for any actual processing or for prediction purposes.

6.2.6.1.2 The SCF Environment

At the DAACs the users will be responsible for submitting the criteria upon which ephemeris and attitude files will be staged for their PGE. The DAACs will populate the Process Control File (PCF) appropriately based on this user supplied criteria. 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 of type binary or HDF. 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|EOSAM1_1995-07-01_12h_01.eph|~/database/sun5/EPH||||5
10501|EOSAM1_1995-07-01_12h_02.eph|~/database/sun5/EPH||||4
10501|TRMM_1994-01-12.eph|~/database/sun5/EPH||||3
10501|TRMM_1994-01-13.eph|~/database/sun5/EPH||||2
10501|TRMM_1994-01-14.eph|~/database/sun5/EPH||||1
10502|EOSAM1_1995-07-01_12h_01.att|~/database/sun5/EPH||||5
10502|EOSAM1_1995-07-01_12h_02.att|~/database/sun5/EPH||||4
10502|TRMM_1994-01-12.att|~/database/sun5/EPH||||3
10502|TRMM_1994-01-13.att|~/database/sun5/EPH||||2
10502|TRMM_1994-01-14.att|~/database/sun5/EPH||||1

or the following if HDF files are used:

10501|EOSAM1_1995-07-01_12h_01.eph.hdf|~/database/sun5/EPH||||5
10501|EOSAM1_1995-07-01_12h_02.eph.hdf|~/database/sun5/EPH||||4
10501|TRMM_1994-01-12.eph.hdf|~/database/sun5/EPH||||3
10501|TRMM_1994-01-13.eph.hdf|~/database/sun5/EPH||||2
10501|TRMM_1994-01-14.eph.hdf|~/database/sun5/EPH||||1
10502|EOSAM1_1995-07-01_12h_01.att.hdf|~/database/sun5/EPH||||5
10502|EOSAM1_1995-07-01_12h_02.att.hdf|~/database/sun5/EPH||||4
10502|TRMM_1994-01-12.att.hdf|~/database/sun5/EPH||||3
10502|TRMM_1994-01-13.att.hdf|~/database/sun5/EPH||||2
10502|TRMM_1994-01-14.att.hdf|~/database/sun5/EPH||||1

See Section 6.2.3 Process Control Tools for a discussion of the PCF and file versioning.

6.2.6.1.3 Running the Orbit/Attitude Simulator

The executable *orbsim* is installed in the $PGSBIN directory at installation time. Make sure the $PGSBIN directory is in your path. To run the program, type "orbsim" at the command line prompt (from any directory).

The simulator is self-explanatory (if you read the messages on the screen). A "q" may be entered at any prompt to quit the simulator. At most prompts there will be a default value that can be selected by merely returning at the prompt without typing any characters. These default values will be indicated by "[]" (e.g., enter a number [7]:).

The first prompt will request the spacecraft ID. The supported values for this are: TRMM, EOS_AM, EOS_PM and EOS_AURA.

The second prompt asks whether HDF files to be generated.

The next prompt will ask users to change orbital elements. Users are given the selection to change the first seven orbital element values. All values should be real numbers, except for the epoch time, which should be in CCSDS ASCII time code. If users do not change orbital elements, the default values will be used. If users change them, the values are overwritten. The fourth prompt will request the start time. Enter the start time in CCSDS ASCII time code (format A or B-see Time and Date Conversion Tools). If users enter only date portion (e.g., 1995-10-20) or date and midnight time (e.g., 1995-10-20T00:00:00), the time starts from midnight. If users enter date and noontime (e.g., 1995-10-20T12:00:00), the time starts from noon. The fifth prompt will request the stop time that should be entered using the same format as the start time. The stop time must be later than the start time. If users only enter date portion, the start and stop time are inclusive (e.g., entering the same start and stop date (e.g., 1995-10-20) will create the spacecraft ephemeris file for that day). The sixth prompt will request the data (or time) interval in seconds. This number is a real number that represents the time interval between data records in the file. These times represent actual ephemeris data. This data will be returned to users directly through PGS_EPH_EphemAttit( ). Ephemeris data requested at times other than the actual record times will be interpolated. The next prompt will ask users to input the time in hour for the data file. The simulator only accepts the divisions of 24 (1, 2, 3, 4, 6, 12, 24). The default value is 24 hours. If users do not enter a value, a whole day data file of 24 hours will be created. Otherwise, the value will be overwritten. Then the simulator will display the start and stop day and time interval entered, as well as the total size (in megabytes) of the data files that will be created. The simulator will then request confirmation of these input values. If the values are rejected the simulator will request the information again beginning with the start day until the values are accepted.

Once the time information has been entered and confirmed the simulator will issue a prompt requesting attitude "noise". This simulator does not allow for any specific yaw, pitch or roll variation, however attitude noise may be introduced to simulate small random variations in the yaw, pitch and roll data reported. At the noise prompt the maximum desired amplitude in arcseconds of the noise should be entered. This should be entered as a real number whose magnitude is LESS than 1000.0 arcseconds (only the magnitude will be considered; the sign of

the number will be ignored). The next prompt will be for attitude rate noise. This should be entered as a real number whose magnitude is LESS than 1000.0 arcseconds/second. Entering “N” at the first prompt (for attitude noise) will turn off this feature; and the roll, pitch and yaw will always be reported as exactly zero. No noise is the default behavior.

The simulator will then prompt for the directory where the ephemeris and attitude files it generates should be written to. The default installation directory is determined from the location of the file leapsec.dat which is assumed to be in $PGSDAT/TD, the simulator will then define the default directory as $PGSDAT/EPH. The location of the output directory is not significant to the tool PGS_EPH_EphemAttit() in any way. The simulator will issue a prompt indicating the default location and asking that the installation directory be specified. Any valid directory may be specified at this prompt (a relative path may be used). The default directory can be selected by merely entering return at this prompt. If an invalid directory is entered the prompt will be reissued until a valid directory is entered.

After a valid directory has been indicated the simulator will attempt to create the spacecraft ephemeris and attitude files for the times requested. The simulator will generate one file each of ephemeris data and attitude data for each date specified. The files generated will follow the naming convention <sc_name>_<date>.eph and <sc_name>_<date>.att for ephemeris and attitude files respectively. The file names and lengths generated by the simulator are for convenience only. Ephemeris and attitude data files may actually have any name and be of any time duration. However, because of the simulator convention of one ephemeris file and one attitude file per day, the simulator will NOT overwrite an existing file for the same spacecraft and the same day, an error message will be issued and the file(s) will be skipped. If for any other reason a file cannot be created the simulator will issue an error message and a prompt asking whether or not it should continue. If directed to continue, the simulator will try one more time to create the file and then continue on to the next file without further warning whether or not the file could be created. The most likely scenario for this is when the user does not have write permission for the directory specified. The above mentioned prompt allows the user to change the directory permission and continue. If the simulator is unable to write to a file that it has already opened (e.g., the disk is full) an error message will be issued.

When all files requested have been written (or skipped), a final prompt is issued allowing the whole process to be repeated.

6.2.6.1.4 Spacecraft Ephemeris and Attitude File Formats

See Appendix L (ECS Spacecraft Ephemeris and Attitude File Formats)

6.2.6.1.5 Tools that Require Spacecraft Ephemeris Files

PGS_EPH_EphemAttit( )
PGS_EPH_GetEphMet( )
PGS_EPH_EphAtt_unInterpolate( )
PGS_EPH_UnInterpEphAtt( )
PGS_CBP_body_inFOV( )
PGS_CBP_Sat_CB_Vector( )

PGS_CSC_GetFOV_Pixel( )
PGS_CSC_SubSatPoint( )
PGS_CSC_Earthpt_FOV( )
PGS_CSC_Earthpt_FixedFOV( )
PGS_CSC_ECItoORB( )
PGS_CSC_ORBtoECI( )
PGS_CSC_ECItoSC( )
PGS_CSC_SCtoECI( )
PGS_CSC_ORBtoSC( )
PGS_CSC_SCtoORB( )

6.2.6.1.6 Warning

The files created by the simulator can be very large and keeping many of them around can quickly fill a hard drive (one day of orbit data for EOS_AM at the default time interval is nearly nine megabytes). The size of the files can be reduced by choosing larger time intervals between data records.

This tool will create files for time in the far future or distant past if the user specifies them. The time of each record in spacecraft ephemeris and attitude files is kept in SDP Toolkit internal time (see Time and Date Conversion Tools) which is a form of TAI time. The user will not be notified if the file created is outside the times for which TAI is defined or currently known (relative to a corresponding UTC time). The simulator will estimate the time and create the file. Such files may contain TAI times on fractional UTC second centers, due to the approximate estimation of TAI-UTC.

6.2.6.2 Ephemeris File Checker

The ECS Spacecraft Ephemeris File Checker can be used to check the format of exiting spacecraft ephemeris files and/or attitude files. This is useful for verifying that an ephemeris file or an attitude file created by a user (i.e., not using the ECS Spacecraft Orbit and Attitude Simulator) is properly formatted. The Ephemeris File Checker is also useful in checking on the time resolution and spacecraft ID of an existing spacecraft ephemeris file or attitude file, as well as in detecting files created without valid leap second data (see Sect. 6.2.6.1.6).

6.2.6.2.1 Brief Description

The spacecraft ephemeris file checker (chkeph) will check the contents of spacecraft ephemeris and attitude files. The checker will read the file header and verify that the metadata contained therein is reasonable. If the header checks out, the checker will then check each record in the file to verify that the times are properly specified (i.e., that the records are properly spaced in time).

6.2.6.2.2 Running the Ephemeris File Checker

The executable *chkeph* is installed in the $PGSBIN directory at installation time. Make sure the $PGSBIN directory is in your path. To run the program type “chkeph” at the prompt with the name(s) of any file(s) to be checked, e.g.,

chkeph TRMM_1998-02-01.eph TRMM_1998-02-02.eph

If the file to be checked is not in the same directory as the one from which chkeph was invoked, the path name must be specified as well (e.g., chkeph ../EPH/TRMM_1998-02-02.eph).

For each file specified chkeph will print out the data contained in the header and check the data records. The first line printed will be the name of the spacecraft and the corresponding numeric value of the Toolkit spacecraft ID (if the spacecraft is an ECS supported s/c). The next two lines will be the numeric start and stop times (respectively) indicated in the header in internal time. Each time will be followed on the same line with the CCSDS ASCII Code (format A) representation of the equivalent UTC time. The next line will be the time interval. Note that this quantity is for record keeping only (i.e., the value has no effect on Toolkit operation). Users creating their own files (i.e., without using the orbsim utility--see above) may set this field to any value. The next line will be the number of records expected to be in the file based on the number of records specified in the file header. The first record will be checked to verify that the time of the record is the same as the time specified as the start time in the file header. Each subsequent record will then be checked to verify that the time of the record is greater than the time of the record immediately preceding it. The last record in the file will be checked to verify that the time of the record is the same as the time specified as the stop time in the file header. The Ephemeris File Checker will issue appropriate error messages if it finds anomalies in the contents of the file that it is checking.

6.2.6.3 Spacecraft Tags Definition File

As of Toolkit 5.2, spacecraft tags are no longer “hard-coded”. Spacecraft tags are defined in an ASCII data file and looked up at runtime. This allows the Toolkit geolocation tools to effectively support any spacecraft that has had it’s ephemeris and attitude data formatted for the Toolkit (see Appendix L. Ephemeris And Attitude File Formats). The spacecraft tags definition file is referenced via the Process Control File with the logical ID of 10801. The file contains a series of records (one per line) of the form:

<sc_tag>,<sc_name>,<eao>

Where:

<sc_tag>	is the numerical (integer) value of the spacecraft tag (passed to Toolkit functions).
<sc_name>	is the actual name of the spacecraft as contained in the ephemeris/attitude file header.
<eao>	is a string consisting of three digits describing the order of the Euler angles (e.g.: 321, 312, 212) as contained in the attitude file.

As delivered the Tookit is configured to support the TRMM, EOS-AM1, EOS-PM and EOS-AURA platforms. These entries in the spacecraft tags file should not be altered. Additional entries may be added below these entries. Each entry should have a unique <sc_name> and <sc_tag>. To ensure backward compatibility, the previous implementation of spacecraft tags has been retained in the Toolkit software. That is, if the tag is TRMM, EOS-AM1, EOS_PM or EOS_AURA and the Spacecraft Tags Definition File is not found, the Toolkit will execute the old “hard coded’ method.

NAME: **PGS_EPH_EphemAttit( )**

SYNOPSIS:

C:

```
#include <PGS_EPH.h>

PGSt_SMF_status
PGS_EPH_EphemAttit(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_boolean        orbFlag,
        PGSt_boolean        attFlag,
        PGSt_integer        qualityFlags[][2],
        PGSt_double         positionECI[][3],
        PGSt_double         velocityECI[][3],
        PGSt_double         eulerAngles[][3],
        PGSt_double         xyzRotRates[][3],
        PGSt_double         attitQuat[][4])
```

FORTRAN:

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

integer function pgs_eph_ephemattit(spacecrafttag,numvalues,asciiutc,
                offsets,orbflag,attflag,qualityflags,
                positioneci,velocityeci,eulerangles,
                xyzrotrates,attitquat)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             orbflag
        integer             attflag
        integer             qualityflags(2,*)
        double precision    positioneci(3,*)
        double precision    velocityeci(3,*)
        double precision    eulerAngles(3,*)
        double precision    xyzrotrates(3,*)
        double precision    attitquat(4,*)
```

DESCRIPTION: This tool gets ephemeris and/or attitude data for the specified spacecraft at the specified times.

INPUTS:

Table 6-75. PGS_EPH_EphemAttit Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A		
numValues	num. Of values requested	N/A		
asciiUTC	UTC time reference start time in CCSDS ASCII time code A format	ASCII	1961-01-01	see NOTES
offsets	array of time offsets in seconds relative to asciiUTC	seconds	depends on asciiUTC	
orbFlag	set to true to get ephemeris data	T/F		
attFlag	set to true to get attitude data	T/F		

OUTPUTS:

Table 6-76. PGS_EPH_EphemAttit Outputs

Name	Description	Units
qualityFlags	quality flags for position and attitude data	see NOTES
positionECI	ECI position	meters
velocityECI	ECI velocity	meters/sec
eulerAngles	s/c attitude as a set of Euler angles	radians
xyzRotRates	angular rates about body x, y and z axes	radians/sec
attitQuat	spacecraft to ECI rotation quaternion	N/A

RETURNS:

Table 6-77. PGS_EPH_EphemAttit Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSEPH_W_BAD_EPHEM_VALUE	One or more values could not be determined
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephemeris/attitude files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephemeris/attitude files could be found for input times
PGSEPH_E_NO_DATA_REQUESTED	Both orbit and attitude flags are set to false
PGSTD_E_SC_TAG_UNKNOWN	Unrecognized/unsupported spacecraft tag
PGSEPH_E_BAD_ARRAY_SIZE	Array size specified is less than 0
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for initial time (asciiUTC)
PGS_E_TOOLKIT	An unexpected error occurred

EXAMPLES:

C:

```
#define ARRAY_SIZE 10

PGSt_double       offsets[ARRAY_SIZE];
PGSt_double       positionECI[ARRAY_SIZE][3];
PGSt_double       velocityECI[ARRAY_SIZE][3];
PGSt_double       eulerAngles[ARRAY_SIZE][3];
PGSt_double       xyzRotRates[ARRAY_SIZE][3];
PGSt_double       attitQuat[ARRAY_SIZE][4];

char              asciiUTC[28];

PGSt_integer      qualityFlags[ARRAY_SIZE][2];

int               I;

PGSt_SMF_status   returnStatus;

** initialize asciiUTC and offsets array **

strcpy(asciiUTC,”1998-02-03T19:23:45.123”);
for (I=0;I<ARRAY_SIZE;I++)
  offsets[I] = (PGSt_double) I;

returnStatus = PGS_EPH_EphemAttit(PGSd_EOS_AM, numValues,
                  asciiUTC, offsets, PGS_TRUE, PGS_TRUE,
                  qualityFlags, positionECI, velocityECI,
                  eulerAngles, xyzRoteRates, attitQuat);

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

FORTRAN:

```
integer           numvalues/10/
integer           I
integer           returnstatus
integer           qualityflags(2,numvalues)

character*27      asciiutc

double precision  offsets(numvalues)
double precision  positioneci(3,numvalues)
double precision  velocityeci(3,numvalues)
double precision  eulerangles(3,numvalues)
```

```
      double precision  xyzrotrates(3,numvalues)
      double precision  attitquat(4,numvalues)

C               initialize asciiutc and offsets array

      asciiutc = '1998-02-03T19:23:45.123'
      do 100 I = 1,numvalues

      offsets(I) = I-1

      returnstatus =  pgs_eph_ephemattit(pgsd_eos_am,numvalues,
     >                  asciiutc,offsets,pgs_true,
     >                  pgs_true,attflag,
     >                  qualityflags,positioneci,
     >                  velocityeci,eulerangles,
     >                  xyzroterates,attitquat)

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

NOTES: **The Euler angles are always relative to the geocentrically based orbital reference frame The attitude rates for TRMM are relative to geodetic orbital reference. The attitude rates for AM1 and later spacecraft are relative to inertial (J2000) reference. In all cases, the attitude rates are the spacecraft angular velocity vector projected on the body axes.**

QUALITY FLAGS:

The quality flags are returned as integer quantities but should be interpreted as bit fields. Only the first 32 bits of each quality flag is meaningfully defined, any additional bits should be ignored (currently integer quantities are 32 bits on most UNIX platforms, but this is not guaranteed to be the case—e.g. an integer is 64 bits on a Cray).

Generally the quality flags are platform specific and are not defined by the Toolkit. Two bits of these flags have, however, been reserved for SDP Toolkit usage. Bit 12 will be set by the Toolkit if no data is available at a requested time, bit 14 will be set by the Toolkit if the data at the requested time has been interpolated (the least significant bit is "bit 0"). Any other bits are platform specific and are the responsibility of the user to interpret. See also Section L.3 (Quality Flags).

See Section 6.2.7.1 (Time Acronyms)

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Appendix L (ECS Spacecraft Ephemeris and Attitude File Formats)

TIME OFFSETS:

This function accepts an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'I' of the input data will be processed at the time: (initial time) + (offset[I]).

Examples:

if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then input[0] will be processed at time 432000.0 and return output[0]

if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then input[0] will be processed at time 432000.0 + offsets[0] and return output[0]

if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0), then each input[I] will be processed at time 432000.0 + offsets[I] and the result will be output[I], where I is on the interval [0,N) ([1,N] in the case of FORTRAN)

ERROR HANDLING:

This function processes data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of this function will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of this function will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of this function will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop

through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status of level of ‘W’ does not necessarily mean that some of the data could not be processed. The ‘W’ level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will be have a status level of ‘W’ to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS above) that processing was not successful at some of the requested times. Therefore in the case of any return status of level ‘W,’ the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

Special Note: for this tool, the associated quality flags will also indicate that no data is available for those points that could not be successfully processed (see QUALITY FLAGS above).

REQUIREMENTS: PGSTK-0720, PGSTK-0141

Get Ephemeris and Attitude Records Without interpolation

NAME: **PGS_EPH_EphAtt_unInterpolate( )**

PGS_EPH_UnInterpEphAtt( )

SYNOPSIS:

C:

```
#include <PGS_EPH.h>

PGSt_SMF_status
PGS_EPH_UnInterpEphAtt(
        PGSt_tag        spacecraftTag,
        char            *asciiUTC_start,
        char            *asciiUTC_stop,
        PGSt_boolean    orbFlag,
        PGSt_boolean    attFlag,
        PGSt_integer    qualityFlag[][2],
        PGSt_integer    numValuesEph,
        PGSt_integer    numValuesAtt,
        char            asciiUTC_Eph[][28],
        char            asciiUTC_Att[][28],
        PGSt_double     positionECI[][3],
        PGSt_double     velocityECI[][3],
        PGSt_double     eulerAngles[][3],
        PGSt_double     xyzRotRates[][3],
        PGSt_double     attitQuat[][4])

PGSt_SMF_status
PGS_EPH_EphAtt_unInterpolate(
        PGSt_tag        spacecraftTag,
        char            *asciiUTC_start,
        char            *asciiUTC_stop,
        PGSt_boolean    orbFlag,
        PGSt_boolean    attFlag,
        PGSt_integer    qualityFlag[][2],
        PGSt_integer    numValues,
        char            asciiUTC_UnAtt[][28],
        PGSt_double     positionECI[][3],
        PGSt_double     velocityECI[][3],
        PGSt_double     eulerAngles[][3],
        PGSt_double     xyzRotRates[][3],
        PGSt_double     attitQuat[][4])
```

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

integer function pgs_eph_uninterpephatt(spacecrafttag, asciitcstart, asciiutcstop, orbflag, attflag, qualityflags, numvalueseph, numvaluesatt,asciiutceph, asciiutcatt, positioneci, velocityeci, eulerangles, xyzrotrates, attitquat)

integer	spacecrafttag
character*27	asciiutcstart
character*27	asciiutcstop
integer	orbflag
integer	attflag
character	asciiutceph(28,*)
character	asciiutcatt(28,*)
integer	numvalueseph
integer	numvaluesatt
integer	qualityflags(2,*)
double precision	positioneci(3,*)
double precision	velocityeci(3,*)
double precision	eulerAngles(3,*)
double precision	xyzrotrates(3,*)
double precision	attitquat(4,*)

integer function pgs_eph_ephatt_uninterpolate(spacecrafttag, asciitcstart, asciiutcstop, orbflag, attflag, qualityflags, numvalues, asciiutcephatt, positioneci, velocityeci, eulerangles, xyzrotrates, attitquat)

integer	spacecrafttag
character*27	asciiutcstart
character*27	asciiutcstop
integer	orbflag
integer	attflag
character	asciiutcephatt(28,*)
integer	numvalues
integer	qualityflags(2,*)
double precision	positioneci(3,*)
double precision	velocityeci(3,*)
double precision	eulerAngles(3,*)
double precision	xyzrotrates(3,*)
double precision	attitquat(4,*)

DESCRIPTION: These tools get actual (without interpolation) ephemeris and/or attitude data records for the specified spacecraft between two specified times. The tool PGS_EPH_EphAtt_unInterpolate() cannot extract both ephemeris and

attitude data records if their numbers are different in the specified time period. Howerver, the tool PGS_EPH_UnInterpEphAtt() which wrapes around PGS_EPH_EphAtt_unInterpolate() can return both records regardless of the difference in number of ephemeris and attitude data records. This tool only will not be able to calculate and return attitude quaternion when the number of ephemeris and attitude data records differ.

INPUTS:

Table 6-78. PGS_EPH_EphAtt_unInterpolate/PGS_EPH_UnInterpEphAtt Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A		
asciiUTC_start	UTC time reference start time in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
asciiUTC_stop	UTC time reference stop time in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
OrbFlag	set to true to get ephemeris data	T/F		
AttFlag	set to true to get attitude data	T/F		
numValues	Max number of expected eph/att records			
numValuesEph	Max number of expected eph records			
numValuesAtt	Max number of expected att records			

OUTPUTS:

Table 6-79. PGS_EPH_EphAtt_unInterpolate/PGS_EPH_UnInterpEphAtt Outputs

Name	Description	Units	Min	Max
numValues	Number of eph/att values between start and stop times			
NumValuesEph	Number of eph values between start and stop times			
numValuesAtt	Number of att values between start and stop times			
asciiUTC_EphAtt	UTC time reference for eph/att records in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
asciiUTC_Eph	UTC time reference for eph records in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
asciiUTC_Att	UTC time reference for att records in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
qualityFlags	quality flags for position and attitude data	See Notes		
positionECI	ECI position	meters		
velocityECI	ECI velocity	meters/sec		
eulerAngles	s/c attitude as a set of Euler angles	radians		
xyzRotRates	angular rates about body x, y and z axes	radian/sec		
AttitQuat	spacecraft to ECI rotation quaternion	N/A		

RETURNS:

Table 6-80. PGS_EPH_EphAtt_unInterpolate/PGS_EPH_UnInterpEphAtt Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSEPH_W_BAD_EPHEM_VALUE	One or more values could not be determined
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephemeris/attitude files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephemeris/attitude files could be found for input times
PGSEPH_E_NO_DATA_REQUESTED	Both orbit and attitude flags are set to false
PGSTD_E_SC_TAG_UNKNOWN	Unrecognized/unsupported spacecraft tag
PGSEPH_E_BAD_ARRAY_SIZE	Array size specified is less than 0
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for initial time (asciiUTC)
PGS_E_TOOLKIT	An unexpected error occurred

EXAMPLES:

C:

```
#define          ARRAY_SIZE          10
PGSt_integer     numValueseph=ARRAY_SIZE;
PGSt_integer     numValuesatt=ARRAY_SIZE;

PGSt_double      positionECI[ARRAY_SIZE][3];
PGSt_double      velocityECI[ARRAY_SIZE][3];
PGSt_double      eulerAngles[ARRAY_SIZE][3];
PGSt_double      xyzRotRates[ARRAY_SIZE][3];
PGSt_double      attitQuat[ARRAY_SIZE][4];
char             asciiUTC_start[28];
char             asciiUTC_stop[28];
char             asciiUTC_Eph[ARRAY_SIZE][28];
char             asciiUTC_Att[ARRAY_SIZE][28];
PGSt_integer     qualityFlags[ARRAY_SIZE][2];
PGSt_SMF_status  returnStatus;
/*initialize asciiUTC start and stop times */
strcpy(asciiUTC_start,"1998-0203T19:23:45.123");
strcpy(asciiUTC_start,"1998-02-03T20:23:45.123");
returnStatus = PGS_EPH_UnInterpEphAtt( PGSd_EOS_AM,
asciiUTC_start, asciiUTC_stop, PGS_TRUE, PGS_TRUE,
qualityFlags, numValueseph, numValuesatt, asciiUTC_Eph,
asciiUTC_Att, positionECI, velocityECI, eulerAngles,
xyzRoteRates, attitQuat);
if (returnStatus != PGS_S_SUCCESS)
{
```

```
                ** do some error handling **
                }
```

FORTRAN:

```
                integer           numvalueseph/10/
                integer           numvaluesatt/10/
                integer           returnstatus
                integer           qualityflags(2,numvalues)
                character*27      asciiutcstart
                character*27      asciiutcstop
                character         asciiutceph(28, numvalues)
                character         asciiutcatt(28, numvalues)
                double precision  positioneci(3,numvalues)
                double precision  velocityeci(3,numvalues)
                double precision  eulerangles(3,numvalues)
                double precision  xyzrotrates(3,numvalues)
                double precision  attitquat(4,numvalues)
C               initialize asciiutc start/stop times
                asciiutcstart = '1998-02-03T19:23:45.123'
                asciiutcstart = '1998-02-03T20:23:45.123'
                returnstatus = pgs_eph_uniterpephatt( pgsd_eos_am,
                asciiutcstart, asciiutcstop, pgs_true, pgs_true,
                qualityFlags, numvalueseph, numvaluesatt,
                asciiutceph,asciiutcatt, positioneci, velocityeci,
                eulerangles, xyzroterates, attitquat)
                if (returnstatus .ne. pgs_s_success) then
                          :
                *** do some error handling ***
                          :
                endif
```

NOTES: **The Euler angles are always relative to the geocentrically based orbital reference frame The attitude rates for TRMM are relative to geodetic orbital reference. The attitude rates for AM1 and later spacecraft are relative to inertial (J2000) reference. In all cases, the attitude rates are the spacecraft angular velocity vector projected on the body axes.**

QUALITY FLAGS:

The quality flags are returned as integer quantities but should be interpreted as bit fields. Only the first 32 bits of each quality flag is meaningfully defined, any additional bits should be ignored (currently integer quantities are 32 bits on most UNIX platforms, but this is not guaranteed to be the case—e.g. an integer is 64 bits on a Cray).

Generally the quality flags are platform specific and are not defined by the Toolkit. Two bits of these flags have, however, been reserved for SDP Toolkit usage. Bit 12 will be set by the Toolkit if no data is available at a requested time, bit 14 will be set by the Toolkit if the data at the requested time has been interpolated (the least significant bit is "bit 0"). Any other bits are platform specific and are the responsibility of the user to interpret. See also Section L.3 (Quality Flags).

See Section 6.2.7.1 (Time Acronyms)

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Appendix L (ECS Spacecraft Ephemeris and Attitude File Formats)

ERROR HANDLING: See notes for PGS_EPH_EphemAttit().

REQUIREMENTS: PGSTK-0720, PGSTK-0141

Get Ephemeris and Attitude Metadata

NAME: **PGS_EPH_GetEphMet( )**

SYNOPSIS:

C:

```
#include <PGS_EPH.h>

PGSt_SMF_status
PGS_EPH_EphMet(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer*       numOrbits,
        PGSt_integer        orbitNumber[],
        char                orbitAscendTime[][28],
        char                orbitDescendTime[][28],
        PGSt_double         orbitDownLongitude[])
```

FORTRAN:

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

integer function pgs_eph_getephmat(spacecrafttag,numvalues,asciiutc,
                offsets,numorbits,orbitnumber,orbitascendtime,
                orbitdescendtime,orbitdownlongitude)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             numorbits
        integer             orbitnumber(*)
        character*27        orbitascendtime(*)
        character*27        orbitdescendtime(*)
        double precision    orbitdownlongitude(*)
```

DESCRIPTION: This tool returns the metadata associated with toolkit spacecraft ephemeris/attitude files.

INPUTS:

Table 6-81. PGS_EPH_GetEphMet Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A		
numValues	num. Of values requested	N/A		
asciiUTC	UTC time reference start time in CCSDS ASCII time code A format	ASCII	1961-01-01	See Notes
offsets	array of time offsets in seconds relative to asciiUTC	seconds	depends on asciiUTC	

OUTPUTS:

Table 6-82. PGS_EPH_GetEphMet Outputs

Name	Description	Units
numOrbits	number of orbits spanned by data set	N/A
orbitNumber	array of orbit numbers spanned by data set	N/A
orbitAscendTime	array of times of spacecraft northward equator crossings	ASCII
orbitDescedTime	array of times of spacecraft southward equator crossings	ASCII
orbitDownLongitude	array of longitudes of spacecraft southward equator crossings	radians

RETURNS:

Table 6-83. PGS_EPH_GetEphMet Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephemeris/attitude files could be found for input times
PGSEPH_E_EPH_BAD_ARRAY_VALUE	Array size specified is less than 0
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_SC_TAG_UNKNOWN	Unrecognized/unsupported spacecraft tag
PGSEPH_W_CORRUPT_METADATA	Same detadata values are believed to be corrupt
PGS_E_TOOLKIT	An unexpected error occured

EXAMPLES:

C:

```
#include <PGS_EPH.h>


#define ORBIT_ARRAY_SIZE 5    /* maximum number of orbits
                                 expected */

#define EPHEM_ARRAY_SIZE 100  /* number of ephemeris data
                                 points */
```

```
PGSt_double     offsets[EPHEM_ARRAY_SIZE];
PGSt_double     orbitdownlongitude[ORBIT_ARRAY_SIZE][3];

PGSt_integer    numOrbits;

PGSt_integer    orbitnumber[ORBIT_ARRAY_SIZE];

char            asciiUTC[28];
char            orbitAscendTime[ORBIT_ARRAY_SIZE][28];
char            orbitDescendTime[ORBIT_ARRAY_SIZE][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<EPHEM_ARRAY_SIZE;i++)
{
    offsets[i] = (PGSt_double) i*60.0;
}

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

      returnStatus = PGS_EPH_GetEphMet(PGSd_EOS_AM,
                                       EPHEM_ARRAY_SIZE,
                                       asciiUTC,
                                       offsets,&numOrbits,
                                       orbitnumber,
                                       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:

```
      implicit none

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

      integer orbit_array_size/1/   ! max. num. orbits expected
      integer ephem_array_size/100/ ! num. of ephem. data points

      double precision offsets(ephem_array_size, 3)
      double precision orbitdownlongitude(orbit_array_size, 3)

      integer          numorbits
      integer          orbitnumber(orbit_array_size)

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

!     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,
     >                                 orbitnumber
     >                                 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: The tool checks for certain kinds of inconsistant or impossible metadata, such as out-of-sequence orbit numbers, orbit start and stop times etc., also see NOTES section of PGS_EPH_EphemAttit( )

REQUIREMENTS: PGSTK-0720, PGSTK-0141

NAME: **PGS_EPH_ManageMasks()**

SYNOPSIS:

C: #include <PGS_EPH.h>

```
PGSt_SMF_status
PGS_EPH_ManageMasks(
PGSt_integer      command,
PGSt_integer      qualityFlagsMasks[2])
```

FORTRAN: include 'PGS_SMF.f'

```
include 'PGS_TD.f'
include 'PGS_EPH_5.f'

integer function
pgs_eph_managemasks(command,qualityflagsmasks)
integer      command
integer      qualityflagsmasks(2)
```

DESCRIPTION: This function is used to get and/or set the values of the ephemeris and attitude quality flags masks. Any bit set in the mask makes the corresponding bit, when encountered in the quality flag from a data packet, fatal.

INPUTS:

Table 6-84. PGS_EPH_ManageMasks Inputs

Name	Description	Units	Min	Max
command	specifies action (get or set) to be taken by this function. Possible value: PGSd_SET and PGSd_GET	N/A	N/A	N/A
qualityFlagsMasks	ephemeris and attitude quality flags masks, in that order	N/A	N/A	N/A

OUTPUTS:

Table 6-85. PGS_EPH_ManageMasks Outputs

Name	Description	Units
qualityFlagsMasks	ephemeris and attitude quality flags masks, in that order.	N/A

RETURNS:

Table 6-86. PGS_EPH_ManageMasks Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSPC_E_DATA_ACCESS_ERROR	Error accessing Process Control File
PGS_E_TOOLKIT	An unexpected error occured

EXAMPLES: The following code would be imbedded in overlying code calling this function. The examples show how to set the flag masks for ephemeris and for attitude data. The other option would be used to get the flag masks from the static buffer in the function itself. To set the masks for an entire run, the PCF can be used. The unit number for the ephemeris mask, PGSd_EPH_QFLAG_MASK is 10507, while that for attitude, PGSd_ATT_QFLAG_MASK is 10508. These equivalences are defined in PGS_EPH.h.

C:

```
PGSt_integer qualFlagM[2]; /* quality flags as integers */


qualFlagM[0]=0x400;  /* rejects “repaired” ephemeris data */
qualFlagM[1]=0x20;   /* rejects attitude data failing red limit */


 returnStatus = PGS_EPH_ManageMasks(PGSd_SET,qualFlagM);
```

FORTRAN:

```
integer    pgs_eph_managemasks
integer*4  flag_value(2)          ! quality flags as integers
integer    setter                 ! to get or set (boolean)
DATA flag_value /1024, 32/        ! rejects repaired ephem. data
```

* and attitude data failing

* red limit

setter = PGSd_SET

returnStatus = pgs_eph_managemasks(setter,flag_value)

NOTES: This function allows for user defined "masks" for the two data quality flags (ephemeris and attitude) associated with spacecraft ephemeris and attitude data. The quality flags are four byte entities (they may be 8 bytes on the Cray but only the first four bytes will be considered) that are interpreted bit by bit for meaning. The least significant bit is bit 0. Currently, the only "fatal" bit (i.e. indicating meaningless data) that will be set prior to access by the toolkit is bit 16. Additionally the toolkit will set bit 12 of the quality flag returned for a given user input time if NO data are found for that input time. Note that this usage is different from most of the other bits, which indicate the state of some existing data point. By default this function will set the mask for each of the quality flags to include bit 16 (fatally flawed data) and bit 12 (no data). This means that any data points returned from the tool PGS_EPH_EphemAttit() with an associated quality flag that has either bit 12 or bit 16 set will be rejected by any TOOLKIT function that makes a call to PGS_EPH_EphemAttit() (note that masking is not applied in the tool PGS_EPH_EphemAttit() itself since users calling this tool directly can examine the quality flags themselves and make their own determination as to which data points to use or reject). The functions affected by using PGS_EPH_ManageMasks() are:

PGS_CBP_Sat_CB_Vector()

PGS_CBP_body_inFOV()

PGS_CSC_ECItoORB()

PGS_CSC_ECItoSC()

PGS_CSC_Earthpt_FOV()

PGS_CSC_Earthpt_FixedFOV()

PGS_CSC_GetFOV_Pixel()

PGS_CSC_ORBtoECI()

PGS_CSC_ORBtoSC()

PGS_CSC_SCtoECI()

PGS_CSC_SCtoORB()

PGS_CSC_SubSatPoint()

For identification of the different bits, please refer to Appendix L of this User Guide.

Users can use this tool or the Process Control File (PCF) to define their own masks which the toolkit will then use instead of the defaults mentioned above. The user defined mask should contain set any bit which the user considers fatal for her/his purpose (e.g. red limit exceeded). WARNING: if the user defined mask does not have bit 16 set, the toolkit will pass through data the associated quality flag of which has bit 16 set. The toolkit will not, however, process any data points if the associated quality flag has bit 12 set (i.e. no data exist) whether or not the user mask has bit 12 explicitly set.

DETAILS: This function will attempt (on its first invocation) to initialize the values of the ephemeris data quality flag masks and the attitude data quality flag masks from values specified in the Process Control File (PCF). If the first call to this function is a "set" (PGSd_SET) operation, the quality flags masks will immediately be set to the input values (i.e. ignoring the values found in the PCF or any errors in attempting to determine the values from the PCF). Once initialized the values of the quality flags masks can then be accessed via the "get" (PGSd_GET) command or altered via the "set" command. The values are retained internally in the function PGS_EPH_ManageMasks().

REQUIREMENTS: PGSTK - 0141, 0720, 0740

6.2.6.3 EPH Functions

PGS_EPH_EphemAttit

See description in 6.2.6.3 Spacecraft Ephemeris and Attitude Tool.

PGS_EPH_GetEphMet

See description in 6.2.6.3 Get Ephemeris and Attitude Metadata.

PGS_EPH_interpolateAttitude

Given a pair of spacecraft attitudes (as Euler angles), attitude rates and their corresponding times this function interpolates the spacecraft attitude and attitude rates to a requested time between the two input times.

PGS_EPH_EphAtt_unInterpolate

Given a pair of spacecraft attitudes (as Euler angles), attitude rates and their corresponding times this function provides the actual data upon requested.

PGS_EPH_interpolatePosVel

Given a pair of spacecraft position vectors, velocity vectors and their corresponding times this function interpolates the spacecraft position and veloctiy to a requested time between the two input times.

6.2.7 Time and Date Conversion Tools

The ability to convert easily and accurately between different representations of time is crucial to EOS science data processing. The time and date conversion routines in the SDP Toolkit will convert between spacecraft time, UTC, International Atomic Time (TAI) and Julian date, as well as converting double precision values to and from CCSDS ASCII formats. Time values are converted for use in science software and as parameters when performing geo-coordinate transformations. In addition, converting time parameters to ASCII or to other more easily read formats facilitates the time values being added to metadata and to various processing logs in a human-readable form.

The spacecraft, UTC, Julian Date, and other times used as input and output for the time and date conversion routines will be in accord with the Consultative Committee for Space Data Systems (CCSDS) standard time code formats where applicable. The formats are described in CCSDS Blue Book, Issue 2, *Time Code Formats*, (CCSDS 301.0-B-2) issued by the Consultative Committee for Space Data Systems (NASA Code- OS, NASA, Washington DC 20546), April 1990. Various EOS supported spacecraft will deliver time data in various CCSDS binary codes. The Toolkit will translate times from these codes to more user friendly formats. Therefore, binary formats will not be described in the present manual. The reader is referred to the Blue Book and to interface documents for the particular spacecraft of interest. The ASCII codes will be described herein both for the convenience of users, and because we have exercised discretion in permitting or forbidding certain truncations.

Because UTC as a real variable is discontinuous at leap seconds boundaries (approximately every one to two years) it has been decided to carry it only in ASCII formats. TAI time runs at the same (Standard International compatible) rate and will be carried as a double precision number, in two ways: Julian Date and seconds from Jan. 1, 1993 UTC midnight.

Toolkit times are either character strings (CCSDS ASCII format), an array of two high precision real values (Toolkit Julian Dates) or a single high precision real value (all other values).

6.2.7.1 Time Acronyms

GAST	Greenwich Apparent Sidereal Time
GMST	Greenwich Mean Sidereal Time
GPS	Global Positioning System
MJD	Modified Julian Date
TAI	International Atomic Time
TDB	Barycentric Dynamical Time
TDT	Terrestrial Dynamical Time
TJD	Truncated Julian Date
UT1	Universal Time
UTC	Coordinated Universal Time

6.2.7.2 ASCII Time Formats

The CCSDS ASCII Time Codes (A and B formats) are defined in the CCSDS Blue Book, pages 2-6 to 2-8. The full format requires all the subfields be present, but certain subsets of the complete time codes are allowed (pages 2-7 to 2-8 of the Blue Book). The Toolkit will handle input and output with slightly different restrictions.

CCSDS ASCII Time Code A as implemented by the Toolkit:

YYYY-MM-DDThh:mm:ss.d->dZ

[Example 2002-02-23T11:04:57.987654Z]

where

YYYY = a four character subfield for year, with value in range 0001-9999

MM = a two character subfield for month with values 01-12, leading zeros required

DD = a two character subfield for day with values in the range 01-eom, where eom is 28, 29, 30, or 31 according to the month (and, for February, the year)

The “T”, a separator, must follow the DD subfield; if and only if there are more characters after the DD subfield; the string will be accepted and parsed such that mm, ss, and d are treated as 0. In that case, a “Z” will still be accepted, but not required, at the end.

hh = a two character subfield for hours, with values 00-23

mm = a two character subfield for minutes, with values 00-59

ss = a two character subfield for seconds, with values 00-59 (00-60 in a positive leap second interval, 00-58 in the case of a negative leap second)

d->d an n-character subfield, (n < 7 for input n = 6 for output), for decimal fraction of a second, with each digit in the range 0-9. If the decimal point appears on input, digits must follow it.

Z - terminator, optional on input

The CCSDS ASCII Time Code B format, described on p. 2-7 of the Blue Book, is:

YYYY-DDDThh:mm:ss.d->dZ

[Example 2002-054T11:04:57.987654Z]

The format is identical to the Code A except that the month, day combination MM-DD is replaced by day of year, i.e.,

DDD = Day of Year as a 3 character subfield with values 001-365 in non leap years and 001-366 in leap years.

NOTE: The CCSDS Formats require all leading zeros be present.

ASCII Time Input

ASCII time input strings may be in either CCSDS ASCII Time Code A format or CCSDS ASCII Time Code B format. All Toolkit functions requiring input ASCII time strings will correctly identify either format.

The Toolkit requires input ASCII time strings to include at least full dates (in format A or B) and will accept ASCII time strings that include times with up to six digits after the decimal point, or subsets truncated from the right (i.e., fractions of a second, whole seconds, minutes, or hours can be omitted by the user and the values will be set to zero. If a subfield is omitted the whole subfield should be omitted; e.g., “ss” cannot be replaced by “s” for seconds.) The time string may also not end with a field delimiter: “T”,”:” or “.”. Users are warned that no error status or message will issue if any of these subfields is missing, so long as truncation is from the right; users should be careful to pass a string of sufficient length to accommodate their data! The Toolkit will not accept truncations from the left; i.e., the year, month and day must be present as four, two, and two digits respectively, or the year as four digits and the day of year as three. Truncation from the left would be too dangerous in view of the coming century change.

Finally, the Toolkit will provide an error message, which will include passing one or more of the offending characters, if the format is violated by input data. In this context, day numbers in excess of the allowable value for the month (and year, for February) are considered errors in format (e.g., a fatal message will issue if DDD = 366 (format B) or MM = 02 and DD = 29 (format A) in a non leap year). A fatal message will issue if the integer part of the seconds subfield runs over 58 in the presence of a negative leap second or over 59 in the absence of a positive leap second. There is no protection against missing data in the presence of a positive leap second if the integer seconds subfield fails to read 60; in that case Toolkit routines cannot populate the leap second interval.

ASCII Time Output

All ASCII time output strings will be in CCSDS ASCII Time Code A format (except for the output of PGS_TD_ASCIItime_AtoB(), which will be in CCSDS ASCII Time Code B format).

The Toolkit will output the full format (date and time), to six digits in the fractional seconds, even though the accuracy may be poorer than one microsecond. There are two reasons why the Toolkit will output microseconds, even though most users will not want numbers more accurate than one millisecond: (i) At least one platform (AM1) plans to provide microseconds; we do not wish to degrade their resolution. (ii) We wish to provide for upgradeability.

The Toolkit will issue a terminal "Z" on the output string to facilitate identification of the end of string and to signify Universal time.

The output strings will be 27 characters in Code A, including the "Z", and 25 in Code B, including the "Z" (Note: this does NOT include the terminating NULL character required in C strings).

6.2.7.3 Toolkit Internal Time (TAI)

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the Toolkit as TAI (upon which it is based). Values are maintained as single high precision real numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The numbers will be negative until midnight, UTC Jan. 1, 1993 and positive after that. The whole number part carries whole seconds and the fractional part carries fractions of a second.

Occasionally, users may wish to relate Toolkit internal time to seconds since Jan. 1, 1958, midnight. The exact numbers' of TAI seconds from 1958-01-01T00:00:00 to 1993-01-01T00:00:00 is 1104537627.0

6.2.7.4 Toolkit Julian Dates

6.2.7.4.1 Format

Toolkit Julian dates are kept as an array of two real high precision numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and that do return a Julian date will return it in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert (internally) the input date to the above format if necessary. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but

will not check and the format of the output may not be what is expected if any other format is used for the input.

6.2.7.4.2 Meaning

Toolkit "Julian dates" are all derived from UTC Julian Dates. A Julian date in any other time stream (e.g., TAI, TDT, UT1, etc.) is the UTC Julian date plus the known difference of the other stream from UTC (differences range in magnitude from 0 seconds to about a minute). Note that although UTC days having leap seconds actually contain 86401 seconds, this is not true for Julian Days of any kind as implemented in the Toolkit. TAI, UT1, TDT and TDB Julian Days are all 86400 seconds, while the UTC Julian Day with the leap second contains duplicate values for one second; only in ASCII form does it have 86401 distinct seconds.

6.2.7.4.3 Examples

In the following examples, all Julian Dates are expressed in Toolkit standard form as two double precision numbers. For display here, the two members of the array are enclosed in braces {} and separated by a comma.

a. UTC to TAI Julian dates conversion

   The Toolkit UTC Julian date for 1994-02-01T12:00:00 is: {2449384.50, 0.5}. TAI-UTC at 1994-02-01T12:00:00 is 28 seconds (.00032407407407 days). The Toolkit TAI Julian date for 1994-02-01T12:00:00 is:

   {2449384.50, 0.5 + .00032407407407} = {2449384.50, 0.50032407407407}

   Note that the Julian day numbers in UTC and the target time stream may be different by + or - 1 for times near midnight.

b. UTC to UT1 Julian dates conversion

   The Toolkit UTC Julian date for 1994-04-10T00:00:00 is: {2449452.50, 0.0}. UT1-UTC at 1994-04-10T00:00:00 is -.04296 seconds (-0.00000049722221 days). The Toolkit UT1 Julian date for 1994-04-10T00:00:00 is:

   {2449452.50, 0.0 - 0.0000004972222}
   = {2449452.50, -0.0000004972222}
   = {2449451.50, 0.9999995027778}

6.2.7.5 Time Boundaries

Many of the Toolkit functions that require time as an input or output keep track of time in the SDP Toolkit internal time format (see above). Most of these functions depend on the file leapsec.dat that contains the values of TAI-UTC (leap seconds).

Some Toolkit functions also (or instead) rely on the file utcpole.dat that contains the values of UT1-UTC.

The times that can be processed by a function may depend on the values maintained in one or both of these files which are updated periodically with new values.

6.2.7.5.1 TAI-UTC Boundaries

The minimum and maximum times that can successfully be processed by functions requiring the value TAI-UTC depend on the file leapsec.dat that relates leap second (TAI-UTC) values to UTC Julian dates. The file leapsec.dat contains dates of new leap seconds and the total leap seconds times on and after Jan 1, 1972. For times between Jan 1, 1961 and Jan 1, 1972 it contains coefficients for an approximation supplied by the International Earth Rotation Service (IERS) and the United States Naval Observatory (USNO). These approximations are the same as adopted by the Jet Propulsion Laboratory (JPL) ephemeris group that produces the DE series of solar system ephemerides, such as DE200, and are used consistently with IERS/USNO/JPL usage. For times after Jan 1, 1961, but before the last date in the file, the Toolkit sets TAI-UTC equal to the total number of leap seconds to date, (or to the USNO/IERS approximation, for dates before Jan 1, 1972). If an input date is before Jan 1, 1961 the Toolkit sets the leap seconds value to 0.0. This is consistent with the fact that, for civil timekeeping since 1972, UTC replaces Greenwich Mean Solar Time (GMT), which had no leap seconds. Thus for times before Jan 1 1961, the user can, for most Toolkit-related purposes, encode Greenwich Mean Solar Time as if it were UTC. If an input date is after the last date in the file, or after Jan 1, 1961, but the file cannot be read, the function will use a calculated value of TAI-UTC based on a linear fit of the data known to be in the table as of early 1997. This value is a crude estimate and may be off by as much as 1.0 or more seconds. If the data file, leapsec.dat, cannot be opened, or the time is outside the range from Jan 1, 1961 to the last date in the file, the return status level will be 'E'. Even when the status level is 'E', processing will continue, using the default value of TAI-UTC (0.0 for times before Jan 1, 1961, or the linear fit for later times). Thus, the user should always carefully check the return status. For times between 1961 and 1972, the leap seconds file contains data used in approximations designed to correct Greenwhich Mean Time to as close an equivalent of UT1 as possible; the Toolkit thus determines Earth rotation from GMT in that period.

6.2.7.5.2 UT1-UTC Boundaries

UT1 is the standard measure of axial Earth rotation and is used by all Toolkit functions for geolocation that locate the spacecraft relative to Earth, or Earth relative to sky (inertial space). UT1 can be reversibly transformed to "Greenwich Hour Angle". It is therefore important to maintain accurate values of UT1. The minimum and maximum times that can successfully be processed by functions requiring the value UT1-UTC depend on the file utcpole.dat that relates UT1-UTC values to UTC dates. The file utcpole.dat starts at June 30, 1972.

The file utcpole.dat, which is maintained periodically, contains final (definitive) and predicted values for UT1 - UTC and related variables that describe polar motion, a small correction (~< 15 meters) to geographic positions due to polar wander and wobble. When the file is updated, the definitive data will reach to within a week in the past of the update time, and the predicted data will extend about one year into the future. A success status message will be returned if all input times correspond to final values. A warning status message will be returned if predicted values

are encountered. An error message will be returned if the time requested is beyond the end of the predictions, or the file cannot be read. The "predicted" values are expected to be satisfactory for most users for several weeks, even if the file is not updated weekly as it should be, because the predictions are rather good for many weeks. Users who desire to reprocess for better accuracy (< 1 m Earth position) will notice their results changing. Because the U.S. Naval Observatory (USNO) gradually refines its older solutions for Earth rotation, which are captured in our file "utcpole.dat", changes at the millimeter to centimeter level may be noticed weeks later even for data processed with "final" values for UT1. (Please note that with Toolkit 5.2 and later, predictions are carried only 83 days ahead, because a leap second could be announced, changing subsequent predictions by one second. Thus the values for 90 days and beyond are no longer relevant; and the error will not exceed about 3.5 m. See section 6.2.7.6.) The following Table, based on error estimates in the USNO data table “finals.data” as of April 23, 1996, indicates the one-sigma errors to be expected in using the file “utcpole.dat” . The days in the left column should be interpreted as days since the last update of the file. The error is due to the inability to predict Earth rotation precisely. The error for times in the recent past (not shown) is only of order < 10 cm. The "interim" data quality supported in TK5 is no longer carried. The first few weeks of predictions are as good as the old "interim" values. Note that the rather small error values in Table 6-84 are a tiny part of the overall difference, UT1 - UTC, which is typically in the range ~ -0.9 to 0.9 seconds, or ~ -420 to +420 m. Please see Appendix N for an example of a utcpole.dat file.

Table 6-87. Estimated Errors in UT1 Predictions (Milliseconds of Time and Equivalent Meters of Geolocation Error)

Prediction Period (Days)	Error (milliseconds) (1 std deviation)	Error (meters at the equator) (1 std deviation)
1	0.3	0.14
30	3.9	1.7
60	6.5	3.0
90	8.8	4.0
120	10.9	4.9
150	12.9	5.8
180	14.8	6.7
225	17.5	7.9
270	20.1	9.0
315	22.5	10.1
360	24.9	11.0
365	25.7	11.5

Because of the reduced accuracy with predicted UT1, and the maximum extension of one year to the predictions, when a relevant function is used, this should carefully check the return status. A success (‘S’) level status message will be returned if all input times correspond to final values. A warning (‘W’) level status message will be returned if any input times correspond to predicted values, even though the error may not be large enough to concern most users. An error (‘E’) level

status message will be returned if the file utcpole.dat cannot be found or if an input time is outside the range of values in the file.

These error messages due to end-of-data could cause problems for users who wish to run simulations one year or more in advance. Users needing to run simulations in the far future can follow procedures shown on the Toolkit Home Page under “Upgrading to Toolkit 5.2” at their own risk. These procedures are risky in an SCF environment or other non-DAAC environment, because of the possibility of pointing at the edited (and hence, false) data files when processing real data. There could also be risk at a DAAC environment if anyone found a way to point at these files with an altered PCF, e.g. if a command-line run were possible in processing science data

6.2.7.6 Updating the Leap Seconds File

The file $PGSDAT/TD/leapsec.dat contains leap seconds data, used by many tools. Since new leap seconds must be appended when they are announced, the file must be periodically updated. The SDP Toolkit contains utilities to perform this update function. If the leap seconds file is more than 83 days old, and the last leap second in the file is also more than 83 days in the past of the time which is being translated by the time tools, an error return will result, because the time cannot be reliably translated. So long as the updates are performed periodically as explained below, users will encounter no problem in processing current or past data, or simulations for the near term future. Users needing to process far future simulations should consult the Toolkit web site or the Toolkit maintenance and operations staff.

The shell script **update_leapsec.sh**, which is found in $PGSBIN, will update the leapsec.dat file to the current date. The Clear Case version, **update_leapsec_CC.sh**, will do the same job within a Clear Case (CM) view. To maintain a current leapsec.dat, the appropriate script must be run at least every month; running once a week offers more protection against an error condition, in case of problems with ftp. The leap seconds are declared by International Earth Rotation Service (IERS) in France, on the basis of their estimates of variations in Earth rotation. Leap seconds are usually added at the start of January or July, and announced nearly six months ahead. The IERS can, however, announce leap seconds on as little as 90 days notice, after which the U.S. Naval Observatory may need up to a week to post them. For that reason, the 83 day file life is enforced, and weekly running of the scripts is advised. Update_leapsec.sh calls PGS_TD_NewLeap, a C program that performs most of the actual update work.

The update is done by collecting the latest information via ftp from the U. S. Naval Observatory. At the DAACs, the process is done automatically by the scheduler. . At Science Computing Facilities, for Toolkits through version 5.2.1, drop 4, users will need to have a ".netrc" file in their home directories, as explained in the comments within the scripts. Later releases will not need such a file.

6.2.7.7 Time and Date Conversion Tools

Convert UTC to TAI Time

NAME: **PGS_TD_UTCtoTAI( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTAI(
        char            asciiUTC[28],
        PGSt_double     *secTAI93);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_utctotai(asciiutc, sectai93)
        character*27        asciiutc
        double precision    sectai93
```

DESCRIPTION: This tool converts UTC time in CCSDS ASCII Time Code (A or B format) to Toolkit internal time (real continuous seconds since 12AM UTC 1-1-93).

INPUTS:

Table 6-88. PGS_TD_UTCtoTAI Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A format or ASCII Time Code B format	time	1961-01-01T00:00:00Z	see NOTES

OUTPUTS:

Table 6-89. PGS_TD_UTCtoTAI Outputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-1009843225.5	see NOTES

RETURNS:

Table 6-90. PGS_TD_UTCtoTAI Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGS_E_TOOKIT	Something unexpected happened, execution aborted

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       secTAI93;

strcpy(asciiUTC,”1993-01-02T00:00:00.000000Z”);
returnStatus = PGS_TD_UTCtoTAI(asciiUTC,&secTAI93);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
         :
         :
}

printf(“TAI: %f\n”,secTAI93);
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotai
integer           returnstatus
character*27      asciiutc
double precision  sectai93

asciiutc = ‘1993-01-02T00:00:00.000000Z’
returnstatus = pgs_td_utctotai(asciiutc,sectai93)
if (returnstatus .ne. pgs_s_success) goto 999
write(6,*) ‘TAI: ‘, sectai93
```

NOTES: **TIME ACRONYMS:**

TAI is: International Atomic Time
UTC is: Universal Coordinated Time

TIME BOUNDARIES:

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210, PGSTK-1220

Convert TAI to UTC Time

NAME: **PGS_TD_TAItoUTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_TAItoUTC(
        PGSt_double   secTAI93,
        char          asciiUTC[28]);
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_taitoutc(sectai93, asciiutc)
        character*27          asciiutc
        double precision      sectai93
```

DESCRIPTION: This tool converts Toolkit internal time (real continuous seconds since 12AM UTC 1-1-93) to UTC time in CCSDS ASCII Time Code A format.

INPUTS:

Table 6-91. PGS_TD_TAItoUTC Inputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-1009843225.577182	see NOTES

OUTPUTS:

Table 6-92. PGS_TD_TAItoUTC Outputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A format	time	1961-01-01T00:00:00	see NOTES

RETURNS:

Table 6-93. PGS_TD_TAItoUTC Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGS_E_TOOLKIT	Something radically wrong occurred

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       secTAI93;
char              asciiUTC[28];

secTAI93 = 86400.;
returnStatus = PGS_TD_TAItoUTC(secTAI93,asciiUTC);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
         :
         :
}

printf("UTC: %s\n",asciiUTC);
```

FORTRAN:

```
implicit none

integer           pgs_td_taitoutc
integer           returnstatus
double precision  sectai93
character*27      asciiutc

sectai93 = 86400.D0
returnstatus = pgs_td_taitoutc(sectai93,asciiutc)
if (returnstatus .ne. pgs_s_success) goto 999
write(6,*) 'UTC: ', asciiutc
```

NOTES: **TIME ACRONYMS:**

TAI is: International Atomic Time
UTC is: Universal Coordinated Time

TIME BOUNDARIES:

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).

REFERENCES FOR TIME:

CCSDS 2301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210, PGSTK-1220

Convert Toolkit Internal Time to TAI Julian Date

NAME: **PGS_TD_TAItoTAIjd()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_double *

PGS_TD_TAItoTAIjd(

        PGSt_double  secTAI93,

        PGSt_double  jdTAI[2])
```

FORTRAN

```
include "PGS_SMF.f"

include "PGS_TD_3.f"

double precision function pgs_td_taitotaijd(sectai93, jdtai)

double precision sectai93

double precision jdtai(2)
```

DESCRIPTION: This function converts time in TAI seconds since 12 AM UTC 1-1-1993 to TAI Julian date.

INPUTS:

Table 6-94. PGS_TD_TAItoTAIjd.c Inputs

Name	Description	Units	Min	Max
secTAI93	Toolkit internal time (seconds since 12 AM	seconds	1958-01-01	none

OUTPUTS:

Table 6-95. PGS_TD_TAItoTAIjd Outputs

Name	Description	Units	Min	Max
jdTAI	TAI Julian date	days	2437300.5	see NOTES

RETURNS: TAI Julian date (address of jdTAI).

EXAMPLES:

C:

```
PGSt_double      secTAI93;

PGSt_double      jdTAI[2];

secTAI93 = 86400.;
```

```
PGS_TD_TAItoTAIjd(secTAI93,jdTAI);

** jdTAI[0] should now have the value: 2448989.5 **
** jdTAI[1] should now have the value: 0.0003125 **
```

FORTRAN:

```
double precision sectai93

double precision jdtai

sectai93 = 86400.D0

call pgs_td_taitotaijd(sectai93, taijd)

! jdtai[0] should now have the value: 2448989.5

! jdtai[1] should now have the value: 0.0003125
```

NOTES: TAI is: Toolkit International Atomic Time measured from 1993-01-01

The translation to and from UTC begins Jan 1, 1961. It is valid until about 6 months after the last leap second, in $PGSDAT/TD/leapsec.dat. When the script $PGSBIN/TD/update_leapsec.sh is run regularly the leap seconds file will be kept current and will be valid six months ahead.

Since TAI was not defined before 1958-01-01 this is the formal lower limit, but practically, the tool will work for any time after 4713 BC, if TAI93 is interpreted as seconds before Jan 1, 1993 UTC midnight.

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK - 1220, 1160, 1170

Convert TAI Julian Date to Toolkit Internal Time

NAME: **PGS_TD_TAIjdtoTAI()**

SYNOPSIS:

C:
```
#include <PGS_TD.h>
PGSt_double
PGS_TD_TAIjdtoTAI(
        PGSt_double  jdTAI[2])
```

FORTRAN:
```
double precision function pgs_td_taijdtotai(jdtai)
double precision jdtai(2)
```

DESCRIPTION: This function converts TAI Julian date to time in TAI seconds since 12 AM UTC 1-1-1993.

INPUTS:

Table 6-96. PGS_TD_TAIjdtoTAI Inputs

Name	Description	Units	Min	Max
jdTAI	TAI Julian date	days	2437300.5	ANY

OUTPUTS: None

RETURNS: Toolkit internal time (seconds since 12 AM UTC 1-1-1993).

EXAMPLES:

```
C     PGSt_double      secTAI93;
      PGSt_double      jdTAI[2];


      jdTAI[0] = 2448989.5;
      jdTAI[1] = 0.0003125;


      secTAI93 = PGS_TD_TAIjdtoTAI(jdTAI);
      /* secTAI93 should now have the value: 86400.*/
```

NOTES: TAI is: International Atomic Time

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK - 1220, 1160, 1170

Convert TAI to GAST

NAME: **PGS_TD_TAItoGAST( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_TAItoGAST(
        PGSt_double   secTAI93,
        PGSt_double   *gast)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f
include 'PGS_TD_3.f'

integer function pgs_td_taitogast(sectai93,gast)
        double precision      sectai93
        double precision      gast
```

DESCRIPTION: This function converts TAI (toolkit internal time) to Greenwich Apparent Sidereal Time (GAST) expressed as the hour angle of the true vernal equinox of date at the Greenwich meridian (in radians).

INPUTS:

Table 6-97. PGS_TD_TAItoGAST Inputs

Name	Description	Units	Min	Max
secTAI93	continuous seconds since 12AM UTC Jan. 1, 1993	seconds	-426297609.0	see NOTES

OUTPUTS:

Table 6-98. PGS_TD_TAItoGAST Outputs

Name	Description	Units	Min	Max
gast	Greenwich Apparent Sidereal Time	radians	0	2PI

RETURNS:

Table 6-99. PGS_TD_TAItoGAST Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_PREDICTED_UT1	Status of UT1-UTC correction is predicted
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSTD_E_NO_UT1_VALUE	No UT1-UTC correction available
PGS_E_TOOLKIT	Something radically wrong occured

EXAMPLES: None

NOTES: **TIME ACRONYMS:**

GAST is: Greenwich Apparent Sidereal Time
TAI is: International Atomic Time

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to in the toolkit as TAI (upon which it is based).See Section 6.2.7.4 Time and Date Conversion Tool Notes

TIME BOUNDARIES:

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:CCSDS 2301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert UTC Time to Spacecraft Clock Time

NAME: **PGS_TD_UTC_to_SCtime( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTC_to_SCtime(
        PGSt_tag        spacecraftTag,
        char            asciiUTC[28],
        PGSt_scTime     scTime[8]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD.f'
include'PGS_TD_3.f'

integer function pgs_td_utc_to_sctime(spacecrafttag, asciiutc, sctime)
        integer         spacecrafttag
        character*27    asciiutc
        character*8     sctime
```

DESCRIPTION: This tool converts UTC in CCSDS Time Code A or B to spacecraft clock time in platform dependent format.

INPUTS: spacecraftTag-Spacecraft identifier; must be one of: PGSd_TRMM, PGSd_EOS_AM, PGSd_EOS_AURA, PGSd_EOS_PM_GIIS, PGSd_EOS_PM_GIRD

asciiUTC-UTC time in CCSDS ASCII Time Code A or CCSDS ASCII Time Code B format. The values of MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information

OUTPUTS: scTime-Spacecraft clock time in platform dependent CCSDS format. UNITS, MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information.

RETURNS:

Table 6-100. PGS_TD_UTCtoSCtime Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSTD_E_SC_TAG_UNKNOWN	Unknown spacecraft tag
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_DATE_OUT_OF_RANGE	Input date is out of range of s/c clock
PGSTD_E_NO_LEAP_SECS	Leap seconds correction unavailable at requested time
PGS_E_TOOLKIT	An unexpected error occurred

EXAMPLES:

C:

```
char              asciiUTC[28];
PGSt_scTime       scTime[8];
PGSt_SMF_status   returnStatus;

strcpy(asciiUTC,”1995-02-04T12:23:44.125438Z”);

returnStatus = PGS_TD_UTC_to_SCtime(PGSd_EOS_AM,asciiUTC,
                                    scTime);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
           :
           :
}
```

FORTRAN:

```
implicit none

integer           pgs_td_utc_to_sctime
character*27      asciiutc
character*8       sctime
integer           returnstatus

asciiutc = ‘1995-02-04t12:23:44.125438Z’

returnstatus = pgs_td_utc_to_sctime(pgsd_eos_am,asciiutc,
                                    sctime)
if (returnstatus .ne. pgs_s_success) then
          :
c         *** do some error handling ***

    :

endif
```

NOTE: WARNING: To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTC VALUE>

Where the proper value of the UTCF should be substituted for <UTC VALUE>.

There is no corresponding problem for AM1 clock time, which is specified to have an accuracy of 100 microseconds.

UTC is: Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

The output spacecraft times vary in format. The supported spacecraft times are in the following formats:

TRMM	CUC (platform specific variant of CCSDS Unsegmented time code(CUC) used)
EOS AM	CDS (platform specific variant of CCSDS day segmented time code (CDS) used)
EOS AURA	CUC
EOS PM GIIS	CDS
EOS PM GIRD	CUC

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK- 1170

Convert Spacecraft Clock Time to UTC Time

NAME: **PGS_TD_SCtime_to_UTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_SCtime_to_UTC(
        PGSt_tag        spacecraftTag,
        PGSt_scTime     scTime[][8],
        PGSt_integer    numValues,
        char            asciiUTC[28],
        PGSt_double     offsets[])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD.f'
include 'PGS_TD_3.f'

integer function pgs_td_sctime_to_utc(spacecrafttag,
                                      sctime,numvalues,asciiutc,
                                      offsets)
        integer             spacecrafttag
        character*8         sctime(*)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
```

DESCRIPTION: This tool converts spacecraft clock time in platform dependent CCSDS format to UTC in CCSDS ASCII Time Code A format.

INPUTS: spacecraftTag-Spacecraft identifier, must be one of: PGSd_TRMM, PGSd_EOS_AM, PGSd_EOS_PM_GIIS, PGSd_EOS_PM_GIRD, PGSd_EOS_AURA

scTime-Array of spacecraft clock times in platform dependent CCSDS format. UNITS, MAX, and MIN depend on the spacecraft, see the files containing the specific conversions for more information.

numValues-number of elements in the input scTime array (and therefore the output offsets array)

OUTPUTS:

Table 6-101. PGS_TD_SCtime_to_UTC Outputs

Name	Description	Units
asciiUTC	UTC time of first s/c clock time in input array (in CCSDS ASCII Time Code A format). The values of MAX, and MIN depend on the spacecraft, add values from prologs!	ASCII
offsets	Array of offsets of each input s/c clock time in input array scTime relative to the first time in the array. This includes the first time as well (i.e., the first offset will be 0.0). The values of MAX, and MIN depend on the first time as well the spacecraft. Add values from prologs!	seconds

RETURNS:

Table 6-102. PGS_TD_SCtime_to_UTC Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSTD_W_BAD_SC_TIME	one or more input s/c times could not be deciphered
PGSTD_E_BAD_INITIAL_TIME	the initial input s/c time (first time in input array) could not be deciphered
PGSTD_E_SC_TAG_UNKNOWN	unknown/unsupported spacecraft ID tag
PGS_E_TOOLKIT	an unexpected error occurred

EXAMPLES:

C:

```
#define ARRAY_SIZE 1000
PGSt_scTime       scTime[ARRAY_SIZE][8];
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE];
PGSt_SMF_status   returnStatus;
*** Initialize scTime array ***
          :
          :
returnStatus = PGS_TD_SCtime_to_UTC(PGSd_EOS_AM,scTime,
                                    ARRAY_SIZE,asciiUTC,
                                    offsets);
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
          :
          :
}
```

FORTRAN:

```
implicit none
integer           pgs_td_sctime_to_utc
integer           array_size
character*8       sctime(array_size)
character*27      asciiutc
double precision  offsets(array_size)
integer           returnstatus
```

```
*** Initialize sctime array ***
            :
            :
returnstatus = pgs_td_sctime_to_utc(pgsd_eos_am,sctime,
                                    array_size,asciiutc,
                                    offsets)
if (returnstatus .ne. pgs_s_success) then
            :
*** do some error handling ***
            :
endif
```

NOTES: WARNING: To properly convert times to or from TRMM s/c clock time the value of the TRMM Universal Time Correlation Factor (UTCF) must be known. This value must be supplied by the user in the Process Control File (PCF). The following line MUST be contained in the PCF for any process that is converting to or from TRMM s/c clock time:

10123|TRMM UTCF value|<UTC VALUE>

Where the proper value of the UTCF should be substituted for <UTC VALUE>.

There is no corresponding problem for AM1 clock time, which is specified to have an accuracy of 100 microseconds.

This function converts an array of input s/c times to an initial time and an array of offsets relative to this initial time. If the first time in the input array cannot be deciphered, this function returns an error. If any other time in the input array cannot be deciphered, the corresponding offset is set to PGSd_GEO_ERROR_VALUE and this function continues after setting the return value to a warning.

See Section 6.2.7.2 (ASCII Time Formats)

The input spacecraft times vary in format. The supported spacecraft times are in the following formats:

TRMM	CUC (platform specific variant of CUC used)
EOS AM	CDS (platform specific variant of CDS used)
EOS AURA	CUC
EOS PM GIIS	CDS
EOS PM GIRD	CUC

UTC: Coordinated Universal Time
TAI: International Atomic Time
CUC: CCSDS Unsegmented Time Code
CDS CCSDS Day Segmented Time Code

REQUIREMENTS: PGSTK-1170

Convert CCSDS ASCII Time Format A to Format B

NAME: **PGS_TD_ASCIItime_AtoB( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_ASCIItime_AtoB(
        char    asciiUTC_A[28],
        char    asciiUTC_B[27]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_asciitime_atob(asciiutc_a,asciiutc_b);
        character*27    asciiutc_a
        character*26    asciiutc_b
```

DESCRIPTION: This Tool converts UTC time in CCSDS ASCII Time Code A to CCSDS ASCII Time Code B.

INPUTS:

Table 6-103. PGS_TD_ASCIItime_AtoB Inputs

Name	Description	Units	Min	Max
asciiUTC_A	UTC Time in CCSDS ASCII Time Code A	N/A	N/A	N/A

OUTPUTS:

Table 6-104. PGS_TD_ASCIItime_AtoB Outputs

Name	Description	Units	Min	Max
asciiUTC_B	UTC Time in CCSDS ASCII Time Code B	N/A	N/A	N/A

RETURNS:

Table 6-105. PGS_TD_ASCIItime_AtoB Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnValue;
char              asciiUTC_A[28];
char              asciiUTC_B[27];

strcpy(asciiUTC_A,”1998-06-30T10:51:28.320000Z”);
returnValue = PGS_TD_ASCIItime_AtoB(asciiUTC_A,asciiUTC_B);
if (returnValue != PGS_S_SUCCESS)
{
** test errors, take appropriate action **
   :
   :
}
printf(“%s\n”,asciiUTC_B);
```

FORTRAN:

```
implicit none

integer           pgs_td_asciitime_atob
integer           returnvalue
character*27      asciiutc_a
character*26      asciiutc_b

asciiutc_a = ‘1998-06-30T10:51:28.320000’
returnvalue = pgs_td_asciitime_atob(asciiutc_a,asciiutc_b)
if (returnvalue .ne. pgs_s_success) goto 999
write(6,*) asciiutc_b
```

NOTES: The output of this tool is in CCSDS ASCII Time Code B format.

See Section 6.2.7.2 (ASCII Time Formats)

REQUIREMENTS: PGSTK-1170, PGSTK-1180, PGSTK-1210

Convert CCSDS ASCII Time Format B to Format A

NAME: **PGS_TD_ASCIItime_BtoA( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_ASCIItime_BtoA(
        char    asciiUTC_B[27],
        char    asciiUTC_A[28]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_asciitime_btoa(asciiutc_b,asciiutc_a);
        character*26   asciiutc_b
        character*27   asciiutc_a
```

DESCRIPTION: This Tool converts UTC time in CCSDS ASCII Time Code B to CCSDS ASCII Time Code A.

INPUTS:

Table 6-106. PGS_TD_ASCIItime_BtoA Inputs

Name	Description	Units	Min	Max
asciiUTC_B	UTC Time in CCSDS ASCII Time Code B	N/A	N/A	N/A

OUTPUTS:

Table 6-107. PGS_TD_ASCIItime_BtoA Outputs

Name	Description	Units	Min	Max
asciiUTC_A	UTC Time in CCSDS ASCII Time Code A	N/A	N/A	N/A

RETURNS:

Table 6-108. PGS_TD_ASCIItime_BtoA Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnValue;
char              asciiUTC_B[27];
char              asciiUTC_A[28];

strcpy(asciiUTC_B,”1998-181T10:51:28.320000Z”);
returnValue = PGS_TD_ASCIItime_BtoA(asciiUTC_B,asciiUTC_A);
if (returnValue != PGS_S_SUCCESS)
{
** test errors, take appropriate action **
      :
      :
}
printf(“%s\n”,asciiUTC_A);
```

FORTRAN:

```
implicit none

integer           pgs_td_asciitime_btoa
integer           returnvalue
character*26      asciiutc_b
character*27      asciiutc_a

asciiutc_b = ‘1998-181T10:51:28.320000’
returnvalue = pgs_td_asciitime_btoa(asciiutc_b,asciiutc_a)
if (returnvalue .ne. pgs_s_success) goto 999
write(6,*) asciiutc_a
```

NOTES: The output of this tool is in CCSDS ASCII Time Code A format.

See Section 6.2.7.2 (ASCII Time Formats)

REQUIREMENTS: PGSTK-1170, PGSTK-1180, PGSTK-1210

Convert UTC to GPS Time

NAME: **PGS_TD_UTCtoGPS( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoGPS(
        char            asciiUTC[28],
        PGSt_double     *secGPS);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_utctogps(asciiUTC,secgps)
        character*27            asciiutc
        double precision        secgps
```

DESCRIPTION: This tool converts from UTC time to GPS time.

INPUTS:

Table 6-109. PGS_TD_UTCtoGPS Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A or B format	time	1961-01-01 T00:00:00	2008-03-30 T23:59:59.999999

OUTPUTS:

Table 6-110. PGS_TD_UTCtoGPS Outputs

Name	Description	Units	Min	Max
secGPS	Continuous real seconds since 0 hrs UTC on Jan. 6, 1980	seconds	-599961636.577182	890956802.999999

RETURNS:

Table 6-111. PGS_TD_UTCtoGPS Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available input time
PGSTD_E_TIME_FMT_ERROR	Error in format of ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of the ASCII UTC time
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
char              asciiUTC[28];
PGSt_double       secGPS;
PGSt_SMF_status   returnStatus;
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus = PGS_TD_UTCtoGPS(asciiUTC,&secGPS);
if(returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus, err, msg);
   printf("\n ERROR:  %s", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_td_utctogps
character*27      asciiutc
double precision  secgps
integer           returnstatus
integer           anerror
character*35      errname
character*150     errmsg

returnstatus = pgs_td_utctogps(asciiutc,secgps)
if(returnstatus .ne. PGS_S_SUCCESS) then
   returnstatus = pgs_smf_getmsg(anerror,errorname,errmsg)
   write(*,*) errname,errmsg
endif
```

NOTES: See Section 6.2.3.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

GPS: Global Positioning System
TAI: International Atomic Time
UTC: Coordinated Universal Time

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert GPS to UTC Time

NAME: **PGS_TD_GPStoUTC( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_Status
PGS_TD_GPStoUTC(
        PGSt_double  secGPS,
        char         asciiUTC[28]);
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_gpstoutc(secgps, asciiutc)
        double precision    secgps
        character*27        asciiutc
```

DESCRIPTION: This tool converts from GPS time to UTC time.

INPUTS:

Table 6-112. PGS_TD_GPStoUTC Inputs

Name	Description	Units	Min	Max
secGPS	Continuous real seconds since 0 hrs UTC on Jan. 6, 1980	seconds	-599961636.577182	see NOTES

OUTPUTS:

Table 6-113. PGS_TD_GPStoUTC Outputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A	time	1961-01-01	see NOTES

RETURNS:

Table 6-114. PGS_TD_GPStoUTC Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap seconds correction for input time
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
char              asciiUTC[28];
PGSt_double       secGPS;
PGSt_SMF_status   returnStatus;
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus = PGS_TD_GPStoUTC(secGPS,asciiUTC);
if(returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus, err, msg);
   printf("\n ERROR:  %s", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_td_gpstoutc
character*27      asciiutc
double precision  secgps
integer           returnstatus
integer           anerror
character*35      errname
character*150     errmsg

returnstatus =  pgs_td_gpstoutc(secgps,asciiUTC)
if(returnstatus .ne. PGS_S_SUCCESS) then
   returnstatus = pgs_smf_getmsg(anerror,errorname,errmsg)
   write(*,*) errname,errmsg
endif
```

NOTES: See Section 6.2.3.2 (ASCII Time Formats)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

GPS: Global Positioning System
TAI: International Atomic Time
UTC: Coordinated Universal Time

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Convert UTC Time to TDT Time

NAME: **PGS_TD_UTCtoTDTjed( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTDTjed(
        char            asciiUTC[28],
        PGSt_double     jedTDT[2]);
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_utctotdtjed(asciiutc, jedtdt)
        character*27            asciiutc
        double precision        jedtdt(2)
```

DESCRIPTION: This tool converts UTC in CCSDS ASCII time format A or B to TDT as a Julian date (TDT = Terrestrial Dynamical Time)

INPUTS:

Table 6-115. PGS_TD_UTCtoTDTjed Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-116. PGS_TD_UTCtoTDTjed Outputs

Name	Description	Units	Min	Max
jedTDT	TDT as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-117. PGS_TD_UTCtoTDTjed Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGSTD_E_NO_LEAP_SECS	Leap second errors
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28] =
                  “2002-06-30T11:04:57.987654Z”;
PGSt_double       jedTDT[2];
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoTDTjed(asciiUTC,jedTDT);
if (returnStatus != PGS_S_SUCCESS)
     {
           PGS_SMF_GetMsg(&returnStatus,err,msg);
           printf(“\nERROR: %s”,msg)
     }
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotdtjed
integer           returnstatus
character*27      asciiutc
double precision  jedtdt(2)
character*33      err
character*241     msg

asciiutc = ‘1998-06-30T10:51:28.320000Z’
returnstatus = pgs_td_utctotdtjed(asciiutc,jedtdt)
if (returnstatus .ne. pgs_s_success)
         returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
         write(*,*) err, msg
endif
```

NOTES:

TIME ACRONYMS:

TDT is: Terrestrial Dynamical Time
UTC is: Coordinated Universal Time

Prior to 1984, there is no distinction between TDT and TDB; either one is denoted “ephemeris time” (ET). Also, the values before 1972 are based on U.S. Naval Observatory estimates, which are the same as adopted by the JPL Ephemeris group that produces the DE series of solar system ephemerides, such as DE200.

Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.2 (ASCII Time Formats)

See See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Convert UTC Time to TDB Time

NAME: **PGS_TD_UTCtoTDBjed( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoTDBjed(
        char            asciiUTC[28],
        PGSt_double     jedTDB[2]);
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_utctotdbjed(asciiutc, jedtdb)
        character*27            asciiutc
        double precision        jedtdb(2)
```

DESCRIPTION: This tool converts UTC in CCSDS ASCII time format A or B to TDB as a Julian date (TDB = Barycentric Dynamical Time)

INPUTS:

Table 6-118. PGS_TD_UTCtoTDBjed Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-119. PGS_TD_UTCtoTDBjed Outputs

Name	Description	Units	Min	Max
jedTDB	TDB as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-120. PGS_TD_UTCtoTDBjed Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGSTD_E_NO_LEAP_SECS	Leap second errors
PGS_E_TOOLKIT	Something unexpected happened, execution of function terminated prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28] =
                  “2002-02-23T11:04:57.987654Z”;
PGSt_double       jedTDB[2];
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoTDBjed(asciiUTC,jedTDB);
if (returnStatus != PGS_S_SUCCESS)
     {
         PGS_SMF_GetMsg(&returnStatus,err,msg);
         printf(“\nERROR: %s”,msg)
     }
```

FORTRAN:

```
implicit none

integer           pgs_td_utctotdbjed
integer           returnstatus
character*27      asciiutc
double precision  jedtdb(2)
character*33      err
character*241     msg

asciiutc = ‘1998-06-30T10:51:28.320000Z’
returnstatus = pgs_td_utctotdbjed(asciiutc,jedtdb)
if (returnstatus .ne. pgs_td_utctotdbjed(asciiutc,jedtdb)
         returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
         write(*,*) err, msg
endif
```

NOTES:

TIME ACRONYMS:

TDB is: Barycentric Dynamical Time
UTC is: Coordinated Universal Time

Prior to 1984, there is no distinction between TDT and TDB; either one is denoted “ephemeris time” (ET). Also, the values before 1972 are based on U.S. Naval Observatory estimates, which are the same as adopted by the JPL Ephemeris group that produces the DE series of solar system ephemerides, such as DE200.

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Compute Elapsed TAI Time

NAME: **PGS_TD_TimeInterval( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

pgs_status
PGS_TD_TimeInterval(
        PGSt_double   startTAI,
        PGSt_double   stopTAI,
        PGSt_double   *interval)
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'

integer function pgs_td_timeinterval( starttai, stoptai, interval)
        double precision      starttai
        double precision      stoptai
        double precision      interval
```

DESCRIPTION: This function computes the elapsed TAI time in seconds between any two time intervals

INPUTS:

Table 6-121. PGS_TD_TimeInterval Inputs

Name	Description	Units	Min	Max
startTAI	start time in TAI	seconds	none	none
stopTAI	stop time in TAI	seconds	none	none

OUTPUTS:

Table 6-122. PGS_TD_TimeInterval Outputs

Name	Description	Units	Min	Max
interval	Elapsed time interval	seconds	none	none

RETURNS:

Table 6-123. PGS_TD_TimeInterval Returns

Return	Description
PGS_S_SUCCESS	Successful return

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       startTAI;
PGSt_double       stopTAI;
PGSt_double       interval;

startTAI = 34523.5;
stopTAI = 67543.2;
returnStatus = PGS_TD_TimeInterval(startTAI,stopTAI,
                                   &interval);
```

FORTRAN:

```
implicit none

integer           pgs_td_timeinterval
integer           returnstatus
double precision  starttai
double precision  stoptai
double precision  interval

returnstatus = pgs_td_timeinterval(starttai,stoptai,
                                   interval)
```

NOTES: This interval is the same as elapsed internal time and is the true interval in System International (SI) seconds.

REQUIREMENTS: PGSTK-1190

Convert UTC in CCSDS ASCII Format to Julian Date Format

NAME: **PGS_TD_UTCtoUTCjd()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUTCjd(
        char            asciiUTC[28],
        PGSt_double     jdUTC[2])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_td_utctoutcjd(asciiutc, jdutc)
character*27            asciiutc
double precision        jdutc(2)
```

DESCRIPTION: Converts ASCII UTC times to UTC Julian Dates

INPUTS:

Table 6-124. PGS_TD_UTCtoUTCjd Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII time Code A or B format	time	1961-01-01	see NOTES

OUTPUTS:

Table 6-125. PGS_TD_UTCtoUTCjd Outputs

Name	Description	Units	Min	Max
jdUTC[2]	UTC Julian date	days	none	none

RETURNS:

Table 6-126. PGS_TD_UTCtoUTCjd Returns

Return	Description
PGS_S_SUCCESS	successful return
PGSTD_M_LEAP_SEC_IGNORED	leap second portion of input time discarded
PGSTD_E_TIME_FMT_ERROR	error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	error in format of input ASCII UTC time
PGS_E_TOOLKIT	something unexpected happened, execution aborted

NOTES: Caution should be used because UTC Julian Date jumps backwards each time a leap second is introduced. Therefore, in a leap second interval the output times will repeat those in the previous second (provided that the UTC ASCII seconds field ran from 60.0 to 60.9999999 etc. as it should during that one second). Therefore, the only known uses for this function are:
(a) to get UT1, (after conversion to modified Julian Date by subtracting 2400000.5) by accessing an appropriate table of differences
(b) to determine the correct Julian Day at which to access any table based on UTC and listed in Julian date, such as leap seconds, UT1, and polar motion tables.

UTC is: Coordinated Universal Time

See section 6.2.7.4 (Toolkit Julian Dates)

REQUIREMENTS: PGSTK - 1170, 1220

Convert UTC Julian Date to CCSDS ASCII Time Code A Format

NAME: **PGS_TD_UTCjdtoUTC()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCjdtoUTC(
        PGSt_double   jdUTC[2],
        PGSt_boolean onLeap,
        char          asciiUTC[28])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer           function      pgs_td_utcjdtoutc(jdutc,onleap,asciiutc)
double precision  jdutc(2)
integer           onleap
character*27      asciiutc
```

DESCRIPTION: This tool converts UTC as a Julian date to UTC in CCSDS ASCII Time Code A format.

INPUTS:

Table 6-127. PGS_TD_UTCjdtoUTC Inputs

Name	Description	Units
jdUTC	UTC time as a Julian date	days
onLeap	Indicates if input time is occurring during a leap second	T/F

OUTPUTS:

Table 6-128. PGS_TD_UTCjdtoUTC Outputs

Name	Description	Units
asciiUTC	UTC time in CCSDS ASCII Time Code A format	time

RETURNS:

Table 6-129. PGS_TD_UTCjdtoUTC Returns

Return	Description
PGS_S_SUCCESS	successful return
PGSTD_E_TIME_FMT_ERROR	a leap second was indicated at an inappropriate time
PGS_E_TOOLKIT	something unexpected happened

EXAMPLES:

C:

```
PGSt_SMF_status  returnStatus;

PGSt_double      jdUTC[2]={2449534.5,0.5};

char             asciiUTC[28];

returnStatus = PGS_TD_UTCjdtoUTC(jdUTC,PGS_FALSE,asciiUTC);

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

/* asciiUTC now contains the value:
   "1994-07-01T12:00:00.000000Z" */

printf("UTC: %s\n",asciiUTC);
```

FORTRAN:

```
integer          pgs_td_utcjdtoutc

integer          returnstatus

double precision jdutc(2)

character*27     asciiutc


jdutc(1) = 2449534.5D0

jdutc(1) = 0.5D0

returnstatus = pgs_td_utcjdtoutc(jdutc,pgs_false,asciiutc)

if (returnstatus .ne. pgs_s_success) goto 999

!  asciiutc now contains the value:
! '1994-07-01T12:00:00.000000Z'

write(6,*) 'UTC: ', asciiutc
```

NOTES: UTC is: Coordinated Universal Time

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems)

Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

See section 6.2.7.4 (Toolkit Julian Dates)

REQUIREMENTS: PGSTK - 1210, 1220, 1160, 1170

Convert UTC to UT1

NAME: **PGS_TD_UTCtoUT1( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUT1(
        char            asciiUTC[28],
        PGSt_double     *secUT1);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_TD_3.f'
include'PGS_CSC_4.f'

integer function pgs_td_utctout1(asciiutc, secut1)
        character*27            asciiutc
        double precision        secut1
```

DESCRIPTION: This tool converts a time from CCSDS ASCII Time (Format A or B) to UT1

INPUTS:

Table 6-130. PGS_TD_UTCtoUT1 Inputs

Name	Description	Units	Min	Max
asciiUTC	UTC time in CCSDS ASCII Time Code A or B format	time	1971-01-01T00:00:00 also see notes	Date

OUTPUTS:

Table 6-131. PGS_TD_UTCtoUT1 Outputs

Name	Description	Units	Min	Max
secUT1	UT1 in seconds from midnight	sec	0.0	86400.999999

RETURNS:

PGS_S_SUCCESS
PGSTD_E_TIME_FMT_ERROR
PGSTD_E_TIME_VALUE_ERROR
PGSCSC_W_PREDICTED_UT1
PGSTD_E_NO_UT1_VALUE
PGS_E_TOOLKIT

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus
char        asciiUTC[28] = “2002-07-27T11:04:57.987654Z
PGSt_double       secUT1
char              err[PGS_SMF_MAX_MNEMONIC SIZE]
char              msg[PGS_SMF_MAX_MSG_SIZE]

returnStatus=PGS_TD_UTCtoUT1(asciiUTC,&secUT1);
if  (returnStatus != PGS_S_SUCCESS)
{
   PGS_SMF_GetMsg(&returnStatus,err,msg);
   printf(“\nERROR: %s”,msg)
}
```

FORTRAN:

```
implicit none

integer           pgs_td_utctout1
integer           returnstatus
character*27      asciiutc
double precision  secut1
character*33      err
character*241     msg

asciiutc = ‘2002-07-27T11:04:57.987654Z’
returnstatus = pgs_td_utctout1(asciiutc,secut1)
if (returnstatus .ne. pgs_s_success) then
   returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
   write(*,*) err, msg
endif
```

NOTES:

Although UT1 was used for civil timekeeping before Jan. 1, 1972, today UT1 is a measure of Earth rotation only; it is a measure of the angle of the Greenwich Meridian from the equinox of date such that 24 hours of System International (SI) seconds (86400 seconds) of TAI or TDT constitute one full revolution. As such, it can be directly reduced to Greenwich Apparent Sidereal Time (GAST). This function should be used with caution near midnight. For example, if UTC is 0.5 seconds before midnight, and UT1 - UTC = 0.6 s, then this function returns 0.1 s, but the day has changed.

Prior to Jan. 1, 1972, either UT1 or, for a brief period, a variant called UT2 that accounts for some of the periodic nonuniformities of Earth rotation, were used for time keeping.

TIME ACRONYMS:

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems), Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1215

Convert UTC to UT1 Julian Date

NAME: **PGS_TD_UTCtoUT1jd( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_UTCtoUT1jd(
        char            asciiUTC[28],
        PGSt_double     jdUT1[2])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'

integer function pgs_td_utctout1jd(asciiutc, jdut1)
        character*27            asciiutc
        double precision        jdut1(2)
```

DESCRIPTION: This tool converts a time from CCSDS ASCII Time (Format A or B) to UT1 Julian date.

INPUTS:

Table 6-132. PGS_TD_UTCtoUT1jd Inputs

Name	Description	Units	Min
asciiUTC	UTC time in CCSDS ASCII Time Code A format or ASCII Time Code B format	ASCII	1961-01-01

OUTPUTS:

Table 6-133. PGS_TD_UTCtoUT1jd Outputs

Name	Description	Units
jdUT1	UT1 Julian date as two real numbers, the first a half integer number of days and the second the fraction of a day between this half integer number of days and the next half integer day number.	days

RETURNS:

Table 6-134. PGS_TD_UTCtoUT1jd Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSTD_M_LEAP_SEC_IGNORED	Leap second portion of input time discarded
PGSTD_E_TIME_FMT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGS_E_TOOLKIT	Something unexpected happened, execution aborted

EXAMPLES: None

NOTES: Although UT1 was used for civil timekeeping before Jan. 1, 1972, today UT1 is a measure of Earth rotation only; it is a measure of the angle of the Greenwich Meridian from the equinox of date such that 24 hours of System International (SI) seconds (86400 seconds) of TAI or TDT constitute one full revolution. As such, it can be directly reduced to Greenwich Apparent Sidereal Time (GAST).

Prior to Jan. 1, 1972, either UT1 or, for a brief period, a variant called UT2 that accounts for some of the periodic nonuniformities of Earth rotation, were used for time keeping.

TIME ACRONYMS:

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.2.7.2 (ASCII Time Formats)

See Section 6.2.7.4 (Toolkit Julian Dates)

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK-1170, PGSTK-1210

Get Leap Second

NAME: **PGS_TD_LeapSec()**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
PGSt_SMF_status
PGS_TD_LeapSec(
        PGSt_double  jdUTC[2],
        PGSt_double  *leapSec,
        PGSt_double  *lastChangeJD,
        PGSt_double  *nextChangeJD,
        char         *leapStatus)
```

FORTRAN

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer funtion pgs_td_leapsec(jdutc,leapsec,lastchangejd,nextchangejd,
                               leapstatus
double precision    jdutc(2)
double precision    leapsec
double precision    lastchangejd
double precision    nextchangejd
character*10        leapstatus
```

DESCRIPTION: This tool accesses the file 'leapsec.dat', extracts the leap second value for an input Julian Day number, and returns an error status.

INPUTS:

Table 6-135. Get Leap Second Inputs

Name	Description	Units	Min	Max
jdUTC	UTC Julian Day number	days (see NOTES)	N/A	N/A

OUTPUTS:

Table 6-136. Get Leap Second Outputs

Name	Description	Units	Min	Max
leapSec	leap second value for day jdUTC, read from table	seconds	0	N/A
lastChangeJD	Julian Day number upon which that leap second value was effective	days (see NOTES)	N/A	N/A
nextChangeJD	Julian Day number of the next ACTUALor PREDICTED leap second	days (see NOTES)	N/A	N/A
leapStatus	indicates whether the leap second value is ACTUAL, PREDICTED, a LINEARFIT, or ZEROLEAPS (leap second value is set to zero if the input time is before the start of the table)	N/A	N/A	N/A

RETURNS:

Table 6-137. Get Leap Seconds Returns

Return	Description
PGS_S_SUCCESS	successful execution
PGSTD_W_JD_OUT_OF_RANGE	invalid input Julian Day number
PGSTD_W_DATA_FILE_MISSING	leap second file not found

EXAMPLES:

```
PGSt_double        jdUTC[2];
PGSt_double        leapsecond;
PGSt_double        lastChangeJD;
PGSt_double        nextChangeJD;

PGSt_SMF_status    returnStatus;
char           leapStatus[10];

jdUTC[0] = 2439999.5;
jdUTC[1] = 0.5;
returnStatus = PGS_TD_LeapSec(jdUTC,&leapsecond,
                              &lastChangeJD,
                              &nextChangeJD,leapStatus);
if (returnStatus != PGS_S_SUCCESS)
{
/*  handle errors  */
}
```

NOTES:

With Toolkit 5.2, the functions that call PGS_TD_LeapSec() will return an error and write a diagnostic message to the Log Status File indicating that an obsoleteformat was encountered in the Leap Seconds file, if they encounter the "PREDICTED" status. "PREDICTED" is no longer supported.

UTC: Coordinated Universal Time

TAI: International Atomic Time

REQUIREMENTS: PGSTK - 1050, 0930

6.2.7.8 TD Functions

PGS_TD_ADEOSIItoTAI

This tool converts ADEOS-II s/c clock time (instrument time + pulse time) to TAI (prototype code).

PGS_TD_ADEOSIItoUTC

This tool converts converts ADEOS-II s/c clock time (instrument time + pulse time) to a UTC string in CCSDS ASCII Time Code A format (prototype code).

PGS_TD_ASCIItime_AtoB

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_ASCIItime_BtoA

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_EOSAMtoTAI

This function converts EOS AM spacecraft clock time in CCSDS day segmented Time Code (CDS) (with implicit P-field) format to TAI (as real continuous seconds since 12AM UTC 1-1-1993).

PGS_TD_EOSAMtoUTC

This function converts EOS AM spacecraft clock time in platform-dependent format to UTC in CCSDS ASCII time code A format.

PGS_TD_EOSAURAGIIStoTAI

This function converts EOS AURA spacecraft GIIS clock time in CCSDS day segmented Time Code (CDS) (with implicit P-field format) to TAI (as real continuous seconds since 12 AM UTC 1-1-1993).

PGS_TD_EOSAURAGIRDtoTAI

This function converts EOS AURA spacecraft GIRD clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format to TAI (as real continuous seconds since 12AM UTC 1-1-1993).

PGS_TD_EOSAURAtoUTC

This function converts EOS AURA spacecraft GIRD clock time in CCSDS unsegmented Time Code (CUC) (with explicit P-field) format to UTC in CCSDS ASCII time code A format.

PGS_TD_EOSPMGIIStoTAI

This function converts EOS PM spacecraft GIIS clock time in CCSDS day segmented Time Code (CDS) (with implicit P-field format) to TAI (as real continuous seconds since 12 AM UTC 1-1-1993).

PGS_TD_EOSPMGIIStoUTC

This function converts EOS PM spacecraft GIIS clock time in platform-dependent format to UTC in CCSDS ASCII time code A format.

PGS_TD_EOSPMGIRDtoTAI

This function converts EOS PM spacecraft GIRD clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format to TAI (as real continuous seconds since 12AM UTC 1-1-1993).

PGS_TD_EOSPMGIRDtoUTC

This function converts EOS PM spacecraft GIRD clock time in CCSDS unsegmented Time Code (CUC) (with explicit P-field) format to UTC in CCSDS ASCII time code A format.

PGS_TD_FGDCtoUTC

This function converts an FGDC ASCII date string and time string to CCSDS ASCII Time Code (format A). The input FGDC time string may be in “Universal Time” or “local time” format.

PGS_TD_GPStoUTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_ISOinttoTAI

This function converts an integer number that represents an ISO time (YYMMDDhh) to TAI.

PGS_TD_ISOinttoUTCjd

This function converts an integer number that represents an ISO time (YYMMDDhh) to a UTC time in toolkit Julian date format.

PGS_TD_JDtoMJD

This function converts a Julian date to a modified Julian date.

PGS_TD_JDtoTJD

This function converts a Julian date to a truncated Julian date.

PGS_TD_JulianDateSplit

This function converts a Julian date to Toolkit Julian date format

PGS_TD_LeapSec

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_MJDtoJD

This function converts a modified Julian date to a Julian date.

PGS_TD_PB5CtoUTCjd

This function converts a time in PB5C time format to TAI (Toolkit internal time).

PGS_TD_PB5toTAI

This function converts a time in PB5 time format to TAI (Toolkit internal time).

PGS_TD_PB5toUTCjd

This function converts a time in PB5 time format to UTC time in toolkit Julian date format.

PGS_TD_SCtime_to_UTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAIjdtoTAI

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAIjdtoTDTjed

This function converts TAI Julian date to TDT Julian ephemeris date.

PGS_TD_TAIjdtoUTCjd

This function converts TAI Julian date to UTC Julian date.

PGS_TD_TAItoGAST

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoISOint

This function converts TAI to an integer number that represents an ISO time (YYMMDDhh).

PGS_TD_TAItoTAIjd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoUDTF

This tool converts TAI to a UDTF integer array.

PGS_TD_TAItoUT1jd

This tool converts continuous seconds since 12AM UTC 1-1-93 to UT1 time as a Julian date.

PGS_TD_TAItoUT1pole

This tool converts continuous seconds since 12AM UTC 1-1-93 to UT1 time as a Julian date and returns x and y polar wander values and UT1-UTC as well.

PGS_TD_TAItoUTC

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_TAItoUTCjd

This tool converts continuous seconds since 12AM UTC 1-1-93 to UTC time as a Julian date.

PGS_TD_TDBjedtoTDTjed

This function converts TDB (Barycentric Dynamical Time) as a Julian ephemeris date to TDT (Terrestrial Dynamical Time) as a Julian ephemeris date.

PGS_TD_TDTjedtoTAIjd

This function converts TDT Julian ephemeris date to TAI Julian date.

PGS_TD_TDTjedtoTDBjed

This function converts TDT (Terrestrial Dynamical Time) as a Julian ephemeris date to TDB (Barycentric Dynamical Time) as a Julian ephemeris date.

PGS_TD_TJDtoJD

This function converts a truncated Julian date to a Julian date.

PGS_TD_TRMMtoTAI

This function converts TRMM spacecraft clock time in CCSDS Unsegmented Time Code (CUC) (with implicit P-field) format to TAI (Toolkit internal time).

PGS_TD_TRMMtoUTC

This function converts TRMM spacecraft clock time in CCSDS unsegmented Time Code (CUC) (with implicit P-field) format to UTC in CCSDS ASCII time code A format.

PGS_TD_TimeInterval

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UDTFtoTAI

This function converts a UDTF integer array to TAI.

PGS_TD_UDTFtoUTCjd

This function converts a UDTF integer array to a UTC Julian date.

PGS_TD_UT1jdtoUTCjd

This tool converts UT1 time as a Julian date to UTC time as a Julian date.

PGS_TD_UTC_to_SCtime

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCjdtoISOint

This function converts a UTC time in toolkit Julian date format to an integer number that represents an ISO time (YYMMDDhh).

PGS_TD_UTCjdtoPB5

This function converts a UTC time in toolkit Julian date format to PB5 time format.

PGS_TD_UTCjdtoPB5C

This function converts a UTC time in toolkit Julian date format to PB5C time format.

PGS_TD_UTCjdtoTAIjd

This tool converts UTC as a Julian date to TAI as a Julian date.

PGS_TD_UTCjdtoUT1jd

This tool converts UTC time as a Julian date to UT1 time as a Julian date.

PGS_TD_UTCjdtoUTC()

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoADEOSII

This function converts UTC in CCSDS ASCII time code A (or B) format to ADEOS s/c clock format (this is a prototype only).

PGS_TD_UTCtoEOSAM

This function converts UTC in CCSDS ASCII time code A (or B) format to EOS AM spacecraft (s/c) clock time in CCSDS Day Segmented (CDS) Time Code (with implicit P-field) format.

PGS_TD_UTCtoEOSAURAGIIS

This function converts UTC in CCSDS ASCII time code A (or B) format to EOS AURA spacecraft GIIS (s/c) clock time in CCSDS Day Segmented (CDS) time code (with implicit P-field) format.

PGS_TD_UTCtoEOSAURAGIRD

This function converts UTC in CCSDS ASCII Time Code A or CCSDS ASCII Time Code B format to EOS AURA spacecraft GIRD clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format.

PGS_TD_UTCtoEOSPMGIIS

This function converts UTC in CCSDS ASCII time code A (or B) format to EOS PM spacecraft GIIS (s/c) clock time in CCSDS Day Segmented (CDS) time code (with implicit P-field) format.

PGS_TD_UTCtoEOSPMGIRD

This function converts UTC in CCSDS ASCII Time Code A or CCSDS ASCII Time Code B format to EOS PM spacecraft GIRD clock time in CCSDS Unsegmented Time Code (CUC) (with explicit P-field) format.

PGS_TD_UTCtoFGDC

This function converts UTC Time in CCSDS ASCII Time Code (format A or B) to the equivalent FGDC ASCII date string and time string. The time string will be in “Universal Time” or “local time” format depending on the value of the input variable tdf.

PGS_TD_UTCtoGPS

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTAI

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTAIjd

This tool converts UTC in CCSDS ASCII time format A or B to TAI as a Julian date.

PGS_TD_UTCtoTDBjed

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTDTjed

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTRMM()

This function converts UTC in CCSDS ASCII time code A (or B) format to TRMM spacecraft (s/c) clock time in CCSDS Unsegmented Time Code (CUC) (with implicit P-field) format.

PGS_TD_UTCtoUT1

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUT1jd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUTCjd

See description in 6.2.7.7 Time and Date Conversion Tools.

PGS_TD_calday

This function converts Julian day to calendar day (year, month, day).

PGS_TD_gast

This function converts GMST, nutation in longitude and TDB Julian date to Greenwich Apparent Sidereal Time expressed as the hour angle of the true vernal equinox of date at the Greenwich meridian (in radians).

PGS_TD_gmst

The function converts UT1 expressed as a Julian day to Greenwich Mean Sidereal Time, i.e. the hour angle of the vernal equinox at the Greenwich meridian (in radians).

PGS_TD_julday

This function converts calendar day (year, month, dat) to Julian day.

PGS_TD_sortArrayIndices

This function sorts an array of PGSt_double (double precision) numbers in ascending order.

PGS_TD_timeCheck

This function accepts a character array (string) as an input and returns a value indicating if the string is in a valid CCSDS ASCII format.

6.3 SDP Toolkit Tools—Optional

6.3.1 Digital Elevation Model Tools

6.3.1.1 DEM Access Tools (HDF-based tools)

The Digital Elevation Model (DEM) access tools described in this section were introduced for the first time in TK 5.2 of the SDP Toolkit. They have been implemented in response to EOS Science Working Group - AM Platform (SWAMP) and ESDIS requests. These tools are meant to replace the access tools described in Section 6.3.1.2 for DEM access. The older access tools and associated ancillary data will continue to be distributed with the Toolkit, as long as there is an identifiable user requirement for them. Please note that the primary ECS production DEMs will supplied in HDF-EOS format and will be accessible through the tools in this section. The older ETOP05 data sets will be accessible in the production system through tools described in Section 6.3.2.

The DEM Toolkit tools in Section 6.3.1.1 are intended for accessing a hierarchy of DEM data sets. In order to utilize these functions, a user must install the SDP Toolkit on their machine. This hierarchy of data sets will include data from multiple resolutions. The DEM tools accesses this information based on resolution; a user indicates from which resolutions they are interested in query data. Each of these resolutions consists of multiple files. For example, the 3 arc second resolution data set (~100 m postings) is divided into 648 (10 degree by 10 degree) files. The number and extent of these files are transparent to the user. The user indicates interest in a particular resolution with a resolution tag. This resolution tag is initialized by the tool PGS_DEM_Open. The resolution tags MUST be initialized, either individually or as an array of the resolution tags, BEFORE any of the other DEM tools may access the data set at that resolution. These initialized resolution tags allow access of the underlying files (in the case of the 3 arc second resolution, the 10 degree by 10 degree files), without having to actually specify the particular physical file.

As mentioned above, the DEM tools may be used with a hierarchy of DEM data sets. Most of the DEM tools not only are able to accept a single resolution tag, but they may even accept a list, an array, of resolution tags. The first element of the array is the tag for the preferred resolution of the data (generally this will be the highest resolution data set). Each successive entry in the array will be in descending interest of use: in general, lower spatial resolution. If one inputs an array of resolution tags to a DEM tool, then one may be able to gain information across resolutions. For example, one may enter an array of resolution tags into the tool PGS_DEM_GetRegion. This tool will go to the data set files of the first resolution tag and extract the region of interest. If any of the points in the region of interest is a fill value, then the tool will access the next data set in the input array (for that particular point). It will continue to step through progressively lower resolution data sets (depending on the order of the elements in the inputted array) until it finds "valid", actual, non fill value, data.

The data sets supported by SDP Toolkit 5.2.20 are the 3 arc second (~100 m postings), 15 arc second (approximately 500m postings), 30 arc second (approximately 1km postings) and 90 arc

second (approximately 3 km postings) resolution data sets. The layer available in all resolutions is elevation (PGSd_DEM_ELEV) and water/land (PGSd_DEM_WATER_LAND). The 15 arc second also includes standard deviation elevation (PGSd_DEM_STDEV_ELEV). Other layers available in both 3 arc second and 30 arc second resolutions are slope (PGSd_DEM_SLOPE), aspect (PGSd_DEM_ASPECT), standard deviation elevation (PGSd_DEM_STDEV_ELEV), and standard deviation slope (PGSd_DEM_STDEV_SLOPE). Also all resolutions include geoid (PGSd_DEM_GEOID). In addition, the 30 arc second data files include quality data such as data source (PGSd_DEM_SOURCE) and quality metric (PGSd_DEM_HORIZONTAL_ACCURACY) and PGSd_DEM_VERTICAL_ACCURACY). All data sets are in HDF-EOS GRID format. The 3 arc second resolution data set is divided into 648 (10 degree by 10 degree) tiles. For each tile there are 2 files, one that includes data for elevation, land/sea mask, slope, aspect, and geoid, and another file that includes data for the standard deviations. Only a few tiles are provided at the 3 arc second resolution, as test data. Full 3 arc data set resides at EDC DAAC. The 15 arc second resolution data set divides the Earth's surface into 24 tiles (2 files per tile as the 3 arc second data set). The 30 arc second resolution data set divides the Earth's surface into 6 tiles (2 files per tile as the 3 arc second data set). The 90 arc second resolution data covers the entire globe in one tile and includes all the ice shelfs for the Antarctica that is in the latest Antarctica version from the Radarsat Antarctica Mapping Program (RAMP). These are delivered with the Toolkit as a provisional data set; updates are possible, for example to replace regions of fill value with actual data. All resolutions are in a Geographic Projection. By geographic, we mean that degrees of latitude and longitude are linearly mapped to row and column pixels, respectively. Please also note that in 15 arc second data real data is not provided for Elevation and Standard Deviation of Elevation for Greenland and Antarctica. The values for these regions are fillvalues. For these regions, we will make new 30 arc second data available with the release 5.2.20 or later of TOOLKIT.

To access these data sets, they must be included in the PCF. The files which make up the 30 arc second resolution should each have a logical ID equal to 10650 for the first file and 10651 for the second file. The logical ID of the 3 arc second resolution files should be 10653 for the first file and 10654 for the second file. The logical ID for the 90 arc second resolution file is 10656. The logical ID of the 15 arc second resolution files should be 10659 for the first file and 10660 for the second file. For more information on setting up a PCF for DEM access, see both the DEM data set README file and the PCF template which accompanies Toolkit 5.2.20.

The DEM access tools described in Section 6.3.1.1 are:

PGS_DEM_Open(): Open the DEM

PGS_DEM_Close(): Close the DEM

PGS_DEM_DataPresent(): Check for Valid DEM Data Point

PGS_DEM_SortModels(): Check for Data in a Specified Region of the DEM

PGS_DEM_GetPoint(): Return Data at Specified DEM Points

PGS_DEM_GetRegion(): Return Data from a Specified Region of the DEM

PGS_DEM_GetMetadata(): Extract Metadata from the DEM

PGS_DEM_GetQualityData(): Access DEM Quality Data

PGS_DEM_GetSize(): Return Size of Specified DEM Region

Open the DEM

NAME: **PGS_DEM_Open()**

SYNOPSIS:

C:

```
#include <PGS_DEM.h>

PGSt_SMF_status
PGS_DEM_Open(
        PGSt_DEM_Tag      resolutionList[],
        PGSt_integer  numResolutions,
        PGSt_integer  layerList[],
        PGSt_integer  numLayers)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_open(resolutionList, numResolutions
                              layerList, numLayers)
    integer       resolutionList(*)
    integer       numResolutions
    integer       layerList(*)
    integer       numLayers
```

DESCRIPTION: This tool initializes a list of resolutions tags which correspond to a series of DEM data sets. These initialized resolution tags are used by the DEM tools. A DEM data set includes all the files of a particular resolution. Presently, only four data sets are available: 3 arc second, 15 arc second, 30 arc second, and 90 arc second resolutions which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_15ARC, PGSd_DEM_30ARC, and PGSd_DEM_90ARC, respectively. A resolution tag MUST be initialized before it may be used in any of the other PGS_DEM tools. Each layer indicated in the layerList will automatically be initialized across all resolutions in the resolutionList. Note that for 90 arc second resolution the only available layers are elevation and Land/Water. For 15 arc second

resolution the only available layers are elevation, Land/Water, and Standard Deviation of Elevation.

INPUTS:

resolutionList[] -- an array of resolution tags, data sets. See Notes.

numResolutions -- the number of resolution tags in the array resolutionList

layerList[] -- the DEM layers to initialize. See Notes.

numLayers -- the number of DEM Layers in the list.

OUTPUTS: N/A

RETURNS: PGS_S_SUCCESS -- success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_integer numLayers;

PGSt_integer layerList[2];

PGSt_integer numResolutions;

PGSt_DEM_Tag resolutionList[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;

resolutionList[1]= PGSd_DEM_30ARC;

numResolutions = 2;

layerList[0] = PGSd_DEM_ELEV;

layerList[1] = PGSd_DEM_WATER_LAND;

numLayers = 2;

/* Open the resolutions and data layer*/

status = PGS_DEM_Open(resolutionList, numResolutions,
layerList, numLayers);

if (status != PGS_S_SUCCESS)

{

/* Do some error handling  ... */
```

FORTRAN:

```
      integer     numLayers
      integer     numResolutions
      integer     layerList(2)
      integer     resolutionList(2)
      integer     status
C INITIALIZE
      resolutionList(1) = PGSd_DEM_3ARC
      resolutionList(2) = PGSd_DEM_30ARC
      layerList(1) = PGSd_DEM_ELEV
      layerList(2) = PGSd_DEM_WATER_LAND
      numResolutions = 2
      numLayers = 2
      status = pgs_dem_open(resolutionList, numResolutions,
     1                      layerList, numLayers)
```

NOTES: resolutionList:

For earlier ECS Deliveries and SCF Toolkits 5.2.2-5.2.7, the data sets that may be inputted are 3 arc second, 30 arc second sets which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_30ARC, respectively. The 15 arc sec data can be handeled with SCF Toolkits 5.2.18 and higher.

layerList:

For ECS Deliveries Drop 4 and later, the only layer that may be inputted for the 3 arc and 30 arc second resolution are elevation, (PGSd_DEM_ELEV), water/land (PGSd_DEM_WATER_LAND), standard deviation elevation (PGSd_DEM_STDEV_ELEV), slope (PGSd_DEM_SLOPE), standard deviation slope (PGSd_DEM_STDEV_SLOPE), and aspect (PGSd_DEM_ASPECT). The other layers that will be available are topographical obscuration (PGSd_DEM_TOP_OBSC). and topographical shadow (PGSd_TOP_SHAD). For 90 arc second resolution the only available layers are elevation, and water/land. For 15 arc second resolution the only available layers are elevation, water/land, and Standard deviation of Elevation..

REQUIREMENTS: PGSTK–0940

Close the DEM

NAME: **PGS_DEM_Close()**

SYNOPSIS:

C:

```
#include <PGS_DEM.h>

PGSt_SMF_status
PGS_DEM_Close(
        PGSt_DEM_Tag      resolutionList[],
        PGSt_integer  numResolutions,
        PGSt_integer  layerList[],
        PGSt_integer  numLayers)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_close(resolutionList, numResolutions
1       layerList, numLayers)
        integer         resolutionList(*)
        integer         numResolutions
        integer         layerList(*)
        integer         numLayers
```

DESCRIPTION: This tool closes the session begun by the tool PGS_DEM_Open. One can close multiple data set sessions simultaneously or independently. If one wants to only close one DEM data set, the array resolutionList should only contain an individual resolution tag. Presently, only four data sets are available: 3 arc second (small test data set), 15 arc second, 30 arc second (provisional global data), and 90 arc second resolutions which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_15ARC , PGSd_DEM_30ARC, and PGS_DEM_90ARC respectively. Each layer in the layerList will automatically be closed across all the resolutions indicated in the resolutionList.

INPUTS: resolutionList[] - an array of resolution tags, data sets. See Notes to PGS_DEM_Open().

numResolutions - the number of resolution tags in the array resolutionList.

layerList[] - the number of DEM Layers to initialize. See Notes to PGS_DEM_Open().

numLayers - the number of DEM Layers in the list.

OUTPUTS: N/A

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_integer numLayers;

PGSt_integer layerList[2];

PGSt_integer numResolutions;

PGSt_DEM_Tag resolutionList[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;

resolutionList[1]= PGSd_DEM_30ARC;

numResolutions = 2;

layerList[0] = PGSd_DEM_ELEV;

numLayers = 1;

/* Close the resolutions and data layer*/

status = PGS_DEM_Close(resolutionList, numResolutions,
layerList, numLayers);
```

FORTRAN:

```
                integer     numLayers
                integer     numResolutions
                integer     layerList(2)
                integer     resolutionList(2)
                integer     status
C INITIALIZE
                resolutionList(1) = PGSd_DEM_3ARC
                resolutionList(2) = PGSd_DEM_30ARC
                layerList(1) = PGSd_DEM_ELEV
                numResolutions = 2
                numLayers = 1
                status = pgs_dem_close(resolutionList, numResolutions,
     1                 layerList, numLayers)
```

REQUIREMENTS: PGSTK–0948

Check for Valid DEM Data Point

NAME: **PGS_DEM_DataPresent()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_DataPresent(
PGSt_DEM_Tag resolution,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double pntLatitude[],
PGSt_double pntLongitude[],
PGSt_integer numPoints,
PGSt_boolean *dataPresent)
```

FORTRAN:

```
include <PGS_SMF.f>
include <PGS_DEM.f>
include <PGS_DEM_14.f>
include <PGS_MEM_7.f>

integer function pgs_dem_datapresent(resolution, layer,
1 positionCode, pntLatitude, pntLongitude, numPoints, dataPresent)
integer resolution
integer layer
integer positionCode
double precision pntLatitude(*)
double precision pntLongitude(*)
integer numPoints
integer dataPresent
```

DESCRIPTION: This tool checks whether pixel(s), at specified latitude(s) and longitude(s), are data or fill values. In dataPresent, either PGS_TRUE or PGS_FALSE will be returned, corresponding to valid data or fill value, respectively.

INPUTS: resolution - the resolution tag for a particular data set. An element of the array resolutionList which is initialized by PGS_DEM_Open. See Notes to PGS_DEM_Open().

layer - indicates which data mask or layers one is accessing. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes.

pntLatitude[ ] and pntLongitude[ ] - the latitude and longitude of the point(s) of interest. See Notes.

numPoints - the number of points to be queried.

OUTPUTS: dataPresent - either PGS_TRUE or PGS_FALSE. PGS_TRUE indicates that a "valid" data value is at the specific location(s). PGS_FALSE indicates that there is a fill value in the extracted data.

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_SMF_status status;
PGSt_integer layer;
PGSt_DEM_Tag resolution;
PGSt_integer numDataPoints;
PGSt_boolean dataPresent;
PGSt_double pntLatitude[3];
PGSt_double pntLongitude[3];

/* initialize input parameters, both resolutions and layer */

resolution= PGSd_DEM_3ARC;

layer = PGSd_DEM_ELEV;


/* initialize the number of data points and pick position of points one
   interested in.  In this case, positions are in signed
   decimal degrees  */
```

```
                numDataPoints = 3;

                pntLatitude[0] =   40.05;

                pntLongitude[0] = -105.3

/*see if selected points have real data*/

                status     =     PGS_DEM_DataPresent(resolution,     layer,
                PGSd_DEM_DEGREE,  pntLatitude,  pntLongitude,  numDataPoints,
                &dataPresent);
```

FORTRAN:

```
                integer     layer

                integer     resolution

                integer status

                integer numDataPoints

                integer dataPresent

                double precision pntLatitude(3)

                double precision pntLongitude(3)

C INITIALIZE resolution and layers

                resolution = PGSd_DEM_3ARC

                layer = PGSd_DEM_ELEV


C INITIALIZE points of interest. in this case, in signed decimal degrees

                numDataPoints = 3

                pntLatitude(1) = 40.04

                pntLongitude(1) = -105.3

                status = pgs_dem_datapresent(resolution, layer,

     1          PGSd_DEM_DEGREE, pntLatitude, pntLongitude,

     1          numDataPoints, dataPresent)
```

NOTES: layer:

See NOTES for layerList of PGS_DEM_OPEN.

positionCode:

The position inputs may be either in signed, decimal degree format or in global pixel format, which correspond to the flags PGSd_DEM_DEGREE and PGSd_DEM_PIXEL, respectively. NOTE: global pixel format is the pixel coordinates of a point in the coordinate system for the whole world. This is NOT the same as pixels inside a single HDF-EOS GRID. The pixel coordinate system is unique for each resolution. The origin of all the pixel coordinate systems is the North, West corner of the globe (180W, 90N). The coordinate system is zero based. The 15 arc second resolution has 86400 pixels spanning from 180 West to 180 East, and 43200 pixels spanning from North Pole to South Pole. The 30 arc second resolution has 43200 pixels spanning from 180 West to 180 East and 21600 pixels spanning from North Pole to South Pole. The 3 arc second resolution has 432000 pixels spanning from 180 West to 180 East and 216000 pixels spanning from North Pole to South Pole. The 90 arc second resolution has 14400 pixels spanning from 180 West to 180 East and 7200 pixels spanning from North Pole to South Pole.

pntLatitude and pntLongitude:

Each longitude point MUST have a corresponding latitude point. The latitude(s) and longitude(s) will be in either signed, decimal degree format or global pixel format, corresponding to the flag indicated by positionCode. If the user is using the flag PGSd_DEM_PIXEL, they should be aware that the values for pntLatitude and pntLongitude will be truncated. In other words, if the user passed in a pixel position which had any decimal information, that information would NOT be used in accessing the data. For example, if the user passed in 1267.34 as a pixel position, it would be truncated to 1267.

REQUIREMENTS: PGSTK–0941

Check for Data in a Specified Region of the DEM

NAME: **PGS_DEM_SortModels()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_SortModels(
PGSt_DEM_Tag resolutionList[],
PGSt_integer  numResolutions,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
PGSt_DEM_Tag *completeDataSet)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_sortmodels(resolutionList,numResolutions,
1 layer, positionCode, latitude, longitude,   completeDataSet)
integer         resolutionList(*)
integer         numResolutions
integer         layer
integer         positionCode
double precision        latitude(2)
double precision        longitude(2)
integer         completeDataSet
```

DESCRIPTION: This tool will check the DEM data sets for complete data in a rectangular region defined by the latitude/longitude pair specified (i.e., upper left hand

corner, lower right hand corner). If there are fill values at any of the points in the defined region, then the tool will query the next resolution tag in the array for that region. The first DEM data set to have complete data in the region of interest will have its corresponding resolution tag returned in completeDataSet. If none of the data sets in the input array is "complete", then the PGSd_DEM_NO_COMPLETE_DATA will be returned.

INPUTS:

resolutionList[] - an array of resolution tags, data sets. See Notes to PGS_DEM_Open().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, latitude and longitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the bounding latitudes and longitudes of the region of interest. See Notes.

OUTPUTS:

completeDataSet - pointer to a resolution tag, data set identifier. The first DEM data set to have complete data in the region of interest will be returned. If none of the resolution tags in the inputted array is "complete", then the PGSd_DEM_NO_COMPLETE_DATA will be returned.

RETURNS:

PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList;

PGSt_integer numResolutions;

PGSt_integer completeData;

PGSt_double latitude[2];

PGSt_double longitude[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;
```

```
                resolutionList[1] = PGSd_DEM_30ARC;

                layer = PGSd_DEM_ELEV;

/* initialize the upper left and lower right corners of the data region..  In
                this case, positions are in signed decimal degrees */

/*upper left corner*/

                latitude[0] =  44.0;

                longitude[0] = -80.0;

/*lower right corner*/

                latitude[1] =  43.0;

                longitude[1] = -79.0;

/* see if region has real data */

                status = PGS_DEM_SortModels(resolutionList, numResolutions,
                layer, PGSd_DEM_DEGREE, latitude, longitude, &completeData);

                if (status!= PGS_S_SUCCESS)

/* Do some error handling ...*/

                else


/*  compare complete data set to the three possibilities to find resolution
                which has complete data across this region.

*/

                if (completeData == PGSd_DEM_3ARC)


/* complete region found in 3 arc second resolution */

                }

                else if (completeData == PGSd_DEM_30ARC)

                {

/* complete region in 30 arc second resolution */

...

                }

                else if (completeData == PGSd_DEM_NO_COMPLETE_DATA)
```

```
                {
/* all resolutions contained fill values within this region */
...
                }
```

FORTRAN:

```
                integer status
                integer layer
                integer resolutionList
                double precision latitude(2)
                double precision longitude(2)
C  initialize input parameters, both resolutions and layer
                resolutionList(1)= PGSd_DEM_3ARC
                resolutionList(2) = PGSd_DEM_30ARC
                layer = PGSd_DEM_ELEV

C  initialize the upper left and lower right corners of the data
C region..  In this case, positions are in signed decimal degrees

C upper left corner
                latitude(1) =  44.0
                longitude(1) = -80.0
C  lower right corner
                latitude(2) =  43.0
                longitude(2) = -79.0

C  see if region has complete data
                status = pgs_dem_sortmodels(resolutionList,
     1          numResolutions, layer, PGSd_DEM_DEGREE, latitude, longitude,
                completeData)
                if (status.NE.PGS_S_SUCCESS) then
```

```
C Do some error handling
                  else

C compare completeData to determine the resolution with complete data
C  in the specified region

                  if (completeData.EQ.PGSd_DEM_3ARC) then

C                       complete data in 3 arc second resolution
....
                  elseif (completeData.EQ.PGSd_DEM_30ARC) then

C                       complete data found in 30 arc second resolution
...
                  elseif (completeData.EQ.PGSd_DEM_NO_COMPLETE_DATA) then

C                 all resolutions contained fill values within this
C                 region
```

NOTES: latitude and longitude:

The first point corresponds to the upper left corner of the rectangular region, and the second point correspond to the lower right corner of the rectangular region. The latitude(s) and longitude(s) will be in either signed, decimal degree format or global pixel format, corresponding to the flag indicated by positionCode. If the user is using the flag PGSd_DEM_PIXEL, she or he should be aware that the values for latitude and longitude will be truncated. In other words, if the user passed in a pixel position which had any decimal information, that information would NOT be used in accessing the data. For example, if the user passed in 1267.34 as a pixel position, it would be truncated to 1267.

REQUIREMENTS: PGSTK–0942

Return Data at Specified DEM Points

NAME: **PGS_DEM_GetPoint()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_GetPoint(
PGSt_DEM_Tag  resolutionList[],
PGSt_integer  numResolutions
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double pntLatitude[],
PGSt_double pntLongitude[],
PGSt_integer numPoints
PGSt_integer interpolation,
void *interpValues)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getpoint(resolutionList, numResolutions, layer, positionCode, pntLatitude, pntLongitude, numPoints, interpolation, interpValue)
integer         resolutionList(*)
integer         numResolutions
integer         layer
integer positionCode
double precision        pntLatitude(*)
double precision        pntLongitude(*)
```

integer numPoints

integer interpolation

‘user defined’ interpValue(*)

DESCRIPTION: This tool attempts to return the data value(s) of the point(s) defined by latitude and longitude. If the latitude and longitude do not exactly correspond to the center (or corner, depending on the manner in which the DEM map has been constructed) of a pixel, the value will be interpolated. Presently, there are only two interpolation methods supported: nearest neighbor and bilinear interpolation. If at this point there is a "hole", a fill value, in the data set, then the tool will access the next resolution tag in the input array. It will continue to step through progressively lower resolution data sets (depending on the order of the elements in the inputted array) until it finds actual data for that point. If all of the DEM data sets have a "hole" at this particular location, then the PGSDEM_M_FILLVALUE_INCLUDED will be returned. Even if some of the queried points are not able to be interpolated (i.e. at the lowest resolution that region is fill value), the value, interpolated value or fill value of the point(s) will be returned in interpValues.

INPUTS: resolutionList - an array of resolution tags, data sets. See Notes to PGS_DEM_SortModels().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

pntLatitude[] and pntLongitude[] - the latitude and longitude of the point of interest. See Notes to PGS_DEM_DataPresent().

numPoints - the number of points to be queried.

interpolation - type of interpolation. see Notes.

OUTPUTS: interpValues - the data value(s) at the designated latitude(s) and longitude(s). See Notes.

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

PGSDEM_M_FILLVALUE_INCLUDED - fill values in the returned data

PGSDEM_M_MULTIPLE_RESOLUTIONS - data accessed from multiple resolutions

EXAMPLES:

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList[2];

PGSt_integer numResolutions;

PGSt_integer numDataPoints;

PGSt_double pntLatitude[10];

PGSt_double pntLongitude[10];

short dataPoints[10)];
```

```
/* NOTE: The type of data buffer should correspond to the type of data that
         one is extracting.  Presently, the only available data are
         PGSd_DEM_ELEV,        PGSd_DEM_SLOPE,        PGSd_DEM_ASPECT,
         PGSd_DEM_STDEV_ELEV,        PGSd_DEM_STDEV_SLOPE,        and
         PGSd_DEM_WATER_LAND which are of type 2 byte, 1 byte, 2
         byte, 2 byte, 2 byte, and 2 byte integers, respectively.  In
         the future, there will be data layers added which are NOT 2
         byte or 1 byte integers.  If one does not know the data type
         of the particular layer, then one should use the tool
         PGS_DEM_GetSize.*/


/* initialize input parameters, both resolutions and layer */

         resolutionList[0]= PGSd_DEM_3ARC;

         resolutionList[1] = PGSd_DEM_30ARC;

         layer = PGSd_DEM_ELEV;


/* initialize the location of the points of interest.  In this case, positions
         are in signed decimal degrees*/

         pntLatitude[0] =  40.05;

         pntLongitude[0] = -105.3;...
```

```
                status = PGS_DEM_GetPoint(resolutionList, numResolutions,
                layer,    PGSd_DEM_DEGREE,    pntLatitude,    pntLongitude,
                numDataPoints,       PGSd_DEM_NEAREST_NEIGHBOR,       (void
                *)dataPoints);


/*Possible status returns*/

                if (status == PGS_S_SUCCESS)

                {

                /*no fill points*/

                ...

                }

                else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

                {

/*fill points included in the extracted data*/

                ...

                }

                else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)

                {

/*no  fill  points  in  data  buffer,  fill  points  interpolated  from  multiple
                resolutions*/

                ...

                }

                else

                {

/*Error in extracting the data */

/* Do some error handling*/
```

FORTRAN:

```
                integer status

                integer layer
```

```
                integer resolutionList(2)
                integer numResolutions
                integer numDataPoints
                double precision pntLatitude(10)
                double precision pntLongitude(10)
                integer*2 dataPoints(10)
C *** NOTE: The type of data buffer should correspond to the type of
C data one is extracting.  Presently, the only available data are
C PGSd_DEM_ELEV, PGSd_DEM_WATER_LAND, PGSd_DEM_SLOPE, PGSd_DEM_ASPECT,
C PGSd_DEM_STD_DEV_ELEV, and PGS_DEM_STDEV_SLOPE which are of type
C 2 byte integers (except for PGSd_DEM_WATER_LAND which is 1 byte
C integer).
C In the future, there will be data layers added which are NOT 2 byte
C or 1 byte integers.  If one does not know the data
C type of the particular
C layer, then one should use the tool PGS_DEM_GetSize.

C initialize input parameters, both resolutions and layer

                resolutionList(1)= PGSd_DEM_3ARC
                resolutionList(2) = PGSd_DEM_30ARC
                layer = PGSd_DEM_ELEV

C initialize points of intereset.  In this case, location is in signed
C decimal  degrees.
                PntLatitude(0) =  40.05
                pntLongitude(0) = -105.3
                status = pgs_dem_getpoint(resolutionList,
     1          numResolutions, layer, PGSd_DEM_DEGREE, pntLatitude,
```

```
     1            pntLongitude, numDataPoints,
     1            PGSd_DEM_NEAREST_NEIGHBOR, dataPoints)

C  Possible status returns

                  if (status .EQ. PGS_S_SUCCESS) then

C  no fill values in this region, in the first resolution
                  ...
                  elseif (status .EQ. PGSDEM_M_FILLVALUE_INCLUDED) then
C  fill values included in extracted data
                  ...
                  elseif (status .EQ. PGSDEM_M_MULTIPLE_RESOLUTIONS) then
C  no fill values included in extracted data.  All fill values
C interpolated  from other resolutions in resolutionList

                  else

C  Error extracting data
C  Do some error handling ...
```

NOTES: All the 15 arc second, 30 arc second, 3 arc second, and 90 arc second DEM data are referenced vertically to mean sea level, which is approximated by the geoid. Thus, the elevation data retrieved by PGS_DEM_GetPoint tool will be with respect to the mean sea level. To get height relative to the WGS84 ellipsoid see note for the function PGS_DEM_GetQualityData.

interpolation:

Presently there is only one type of interpolation, nearest neighbor, PGSd_DEM_NEAREST_NEIGHBOR, and bilinear interpolation, PGSd_DEM_BILINEAR.

interpValues:

If the function locates fill values in the extracted data from the first resolution in the resolutionList, it will attempt to interpolate from the other resolutions. If the point of interest corresponds to a fill value at the lowest resolution (the last resolution tag of resolutionList), then this fill value(s) will be returned.

The land/water classes are described below:

0. Shallow Ocean (Ocean <5k from coast OR <50m deep; i.e., a buffer zone around all coastal areas and islands, plus shallow areas up to 50m deep that are further than 5km from the land). Includes the appropriate parts of the Black Sea, Red Sea, Mediterranean Sea, Hudson Bay, and other ocean-connected seas.

1. Land (not anything else).

2. Ocean Coastlines and Lake Shorelines (an actual boundary line).

3. Shallow Inland Water (Inland Water <5km from shore OR <50m deep; i.e., a buffer zone around all lake shores and inland islands, plus shallow areas up to 50m deep that are further than 5km from the land). Includes the appropriate parts of the Caspian Sea, Aral Sea, Great Lakes, "2-line" rivers, etc.

4. Ephemeral (intermittent) Water (from Digital Chart of the World).

5. Deep Inland Water (Inland water >5km from shoreline AND >50m deep; i.e., Lake waters beyond 5km from their shore or islands, and greater than 50m eep). Includes the appropriate parts of the Caspian Sea, Aral Sea, Great akes, etc.

6. Continental Shelf Ocean (Ocean >5km from coast AND between 50m and 500m deep); i.e., Oceans beyond 5km from coastal areas and islands, and greater than 50m deep but less than 500m deep. Primarily represents the Continental shelf areas.

7. Deep Ocean (Ocean >5km from coast AND >500m deep); i.e., The really deep oceans.

IMPORTANT!! It is the user's responsibility to allocate the appropriate amount of space for interpValue. Note, that each mask has its own data type, see PGS_DEM_GetSize.

WARNING: Because of memory limitations it is not possible to extract more than a certain number of points by a single call to this function. The maximum number of points that can be extracted by one call to this function depends on the machine configuration at the runtime.

REQUIREMENTS: PGSTK–0943

Return Data from a Specified Region of the DEM

NAME: **PGS_DEM_GetRegion()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_GetRegion(
PGSt_DEM_Tag  resolutionList[],
PGSt_integer numResolutions,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_integer interpolation,
PGSt_double latitude[2],
PGSt_double longitude[2],
void *dataRegion,
PGSt_double regionSize[2],
PGSt_double firstElement[2],
PGSt_double pixelSize[2])
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getregion(resolutionList, numResolutions, layer, positionCode, interpolation, latitude, longitude, dataRegion, regionSize, firstElement, pixelSize)
integer         resolutionList(*)
integer         numResolutions
integer         layer
integer         positionCode
```

integer	interpolation
double precision	latitude(2)
double precision	longitude(2)
‘user defined’	dataRegion(*)
double precision	regionSize(2)
double precision	firstElement(2)
double precision	pixelSize(2)

DESCRIPTION: This tool returns the data from a rectangular region of the DEM data set. In addition to returning an array of data, this tool will return the dimension of the region in terms of coordinate degrees, the coordinates of the first element of the dataRegion, and the size of the pixel. If any of the points in the region of interest is a "hole", a fill value, then the tool will access the next DEM data set in the input array. It will continue to step through progressively lower resolution data sets (depending on the order of the resolution tags in the inputted array) until it finds "valid", actual data. If all of the inputted resolutions have a "hole" at these specific locations, then the PGSDEM_M_FILLVALUE_INCLUDED will be returned. Even if some of the queried points are not able to be interpolated (i.e., at the lowest resolution that region is fill value), the data region is still returned. The only consequence is that dataRegion will not consist solely of "valid" and interpolated data but will also contain fill values

INPUTS: resolutionList - an array of resolution tags, data sets. See Notes to PGS_DEM_SortModels().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask or layer one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, latitude and longitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the bounding latitudes and longitudes of the region of interest. See Notes to PGS_DEM_SortModels().

interpolation - type of interpolation. See Notes to PGS_DEM_GetPoint().

OUTPUTS: dataRegion - an array in which the DEM data will be returned. See Notes.

regionSize[2] - an array indicating the size of the region in terms of the degrees of latitude and longitude. The array elements correspond to latitude and longitude respectively. The values will be in decimal format.

firstElement[2] - an array indicating the latitude and longitude, in decimal degree format, of the first element. The elements of the array correspond to latitude and longitude respectively.

pixelSize[2] - an array indicating the size of a pixel in terms of the degrees of latitude and longitude. The array elements correspond to latitude and longitude respectively.

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

PGSDEM_M_FILLVALUE_INCLUDED - fill values in the returned data

PGSDEM_M_MULTIPLE_RESOLUTIONS - data accessed from multiple resolutions

EXAMPLES:

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList[2];

PGSt_integer numResolutions;

PGSt_double latitude[2];

PGSt_double longitude[2];

PGSt_double regionSize[2];

PGSt_double firstElement[2];

PGSt_double pixelSize[2];

short * dataRegion;

/* NOTE: The type of data buffer should correspond to the type of data one is
         extracting.  Presently, the only available data are
         PGSd_DEM_ELEV, PGSd_DEM_WATER_LAND, PGSd_DEM_SLOPE,
         PGSd_DEM_ASPECT, PGSd_DEM_STD_DEV_ELEV, and
         PGS_DEM_STDEV_SLOPE which are of type 2 byte integers
         (except for PGSd_DEM_WATER_LAND which is 1 byte integer).

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;
```

```
                resolutionList[1] = PGSd_DEM_30ARC;

                layer = PGSd_DEM_ELEV;


/* initialize the location of the region of interest.  In this case, positions
                are in signed decimal degrees*/


/* upper left corner of region */

                pntLatitude[0] =  44.05;

                pntLongitude[0] =  -80.0;

/* lower right corner of region */

                latitude[1] = 43.0;

                longitude[1] = -78.8;


/*Allocate space for the buffers GetRegion.  It is the USER's RESPONSIBILITY
                TO ALLOCATE SPACE. .  If one does not know the data type or
                the extent of one's region in global pixels, then one should
                use the tool PGS_DEM_GetSize. */

                status  =  PGS_DEM_GetRegion(resolutionList,  numResolutions,
                layer ,PGSd_DEM_DEGREE, PGSd_DEM_NEAREST_NEIGHBOR, latitude,
                longitude, dataRegion, regionSize, firstElement, pixelSize);


/* possible status returns */

                if (status == PGS_S_SUCCESS)

                {

                /*no fill points*/

                ...

                }

                else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

                {

/*fill points included in the extracted data*/

                ...
```

```
                    }
                    else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)
                    {
/*no fill points in data buffer, fill points interpolated from multiple
                    resolutions*/
                    ...
                    }
                    else
                    {
/*Error in extracting the data */
/* Do some error handling*/
```

FORTRAN:

```
                    integer status
                    integer layer
                    integer resolutionList(2)
                    integer numResolutions
                    double precision latitude(2)
                    double precision longitude(2)
                    double precision regionSize(2)
                    double precision firstElement(2)
                    double precision pixelSize(2)
                    integer*2 dataRegion(*)
C *** NOTE: The type of data buffer should correspond to the type of data one
                    is extracting.
C Presently, the only available data are PGSd_DEM_ELEV, PGSd_DEM_WATER_LAND,
C PGSd_DEM_SLOPE, PGSd_DEM_ASPECT, PGSd_DEM_STD_DEV_ELEV,
C and PGS_DEM_STDEV_SLOPE which are of type 2 byte integers (except
C for PGSd_DEM_WATER_LAND which is 1 byte integer).
```

```
C In C the future, there will be data layers added which are NOT 2 byte or 1
byte integers. It is

C the USER's RESPONSIBILITY TO ALLOCATE SPACE.  If one does not know the data
type C or C the extent of one's region in global pixels, then one should use
the tool C PGS_DEM_GetSize. ***


C   initialize input parameters, both resolutions and layer

                resolutionList(1)= PGSd_DEM_3ARC

                resolutionList(2) = PGSd_DEM_30ARC

                      layer = PGSd_DEM_ELEV


C  initialize the region of interest. In this case, the position is in signed
                decimal degrees.

C  upper left corner of region

                pntLatitude(1) =  44.05

                pntLongitude(1) =  -80.0

C   lower right corner of region

                latitude(2) = 43.0

                longitude(2) = -78.8

                status = PGS_DEM_GetRegion(resolutionList,

     1          numResolutions, layer ,PGSd_DEM_DEGREE,

     1          PGSd_DEM_NEAREST_NEIGHBOR, latitude, longitude,

     1          dataRegion, regionSize, firstElement, pixelSize)


C  possible status returns


                if (status == PGS_S_SUCCESS)

C  **no fill points

                else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

C **fill points included in extracted data
```

```
                  else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)
C  **no fill points in extracted data.  All fill points
C interpolated from other resolutions in resolutionList **
                  else
C  **Error extracting data
C  **Do some error handling ...
```

NOTES: dataRegion:

If the function locates fill values in the extracted data from the first resolution in the resolutionList, it will attempt to interpolate from the other resolutions. If the point of interest corresponds to a fill value at the lowest resolution (the last resolution tag of resolutionList), then this fill value(s) will be returned.

IMPORTANT!! It is the user's responsibility to allocate the appropriate amount of space for dataRegion. Note, that each mask has its own data type, see PGS_DEM_GetSize.

REQUIREMENTS: PGSTK–0944

Extract Metadata from the DEM

NAME: **PGS_DEM_GetMetadata()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_GetMetadata(
PGSt_DEM_Tag resolution.
PGSt_integer layer,
PGSt_double pixLatInfo[2],
PGSt_double pixLonInfo[2],
char *positionUnits,
PGSt_double *scaling,
PGSt_double *offset,
PGSt_double *fillValue,
char *dataUnits,
PGSt_integer *mapProjection,
PGSt_boolean *qualityAssurLayer)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getmetadata(resolution, layer, pixLatInfo, pixLonInfo, positionUnits, scaling, offset, fillValue, dataUnits, mapProjection, qualityAssurLayer)
integer         resolution
integer         layer
double precision        pixLatInfo(2)
double precision        pixLonInfo(2)
character       positionUnits(*)
```

double precision scaling

double precision offset

double precision fillValue

character dataUnits(*)

integer mapProjection

integer qualityAssurLayer

DESCRIPTION: This tool accesses the general metadata that pertains to a single DEM data set the metadata is for the whole data set, not for isolated geographic sections of the data. Some of the metadata are valid for all the attributes, but other metadata will be mask specific.

INPUTS: resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

layer - indicates which data mask or layer one is accessing. See Notes to PGS_DEM_DataPresent().

OUTPUTS:

pixLatInfo - an array of information on the global row pixels. See Notes.

pixLonInfo - an array of information on the global column pixels. See Notes.

positionUnits - units of the position coordinates

scaling - a pointer to the scaling factor to convert attribute data to its appropriate units

offset - a pointer to an offset to convert the attribute data (after scaling) to a meaningful value

resolution - a pointer to the resolution of the attribute data

dataUnits - the units of the attribute data

fillValue - a pointer to the fill value of the specified attribute data

mapProjection - a pointer to the type of geographic projection applied to the attribute data. Corresponds to different projection flags. See HDF-EOS User's Guide for projection codes.

qualityAssurLayer - flag indicating a quality assurance and source layer for the attribute data. This will either have the value PGS_TRUE or PGS_FALSE which corresponds to the existence and the absence, respectively, of a quality assurance layer.

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
                PGSt_SMF_status status;

                PGSt_integer layer;

                PGSt_DEM_Tag resolution;

                PGSt_double pixLatInfo[2];

                PGSt_double pixLonInfo[2];

                PGSt_double scaling;

                PGSt_double offset;

                PGSt_double fillValue;

                character *positionUnits;

                character *dataUnits;

                PGSt_integer mapProjection;

                PGSt_boolean qualityAssuranceLayer;

/* initialize resolution and layer*/

                resolution = PGSd_DEM_3ARC;

                layer = PGSd_DEM_ELEV;

/* allocate enough space for positionUnits and dataUnits string *

                positionUnits = calloc(30. sizeof(char));

                dataUnits = calloc(30. sizeof(char));

                status = PGS_DEM_GetMetadata(resolution, layer, pixLatInfo,
                pixLonInfo, positionUnits, &scaling, &offset, &fillValue,
                dataUnits, &mapProjection, &qualityAssuranceLayer);

                if (status != PGS_S_SUCCESS)

                {

/* Do some error handling  */
```

FORTRAN:

```
                integer status
                integer layer
                integer resolution
                double precision pixLatInfo(2)
                double precision pixLonInfo(2)
                double precision scaling
                double precision offset
                double precision fillValue
                integer mapProjection
                integer qualityAssuranceLayer
                character positionUnits(30)
                character dataUnits(30)
c   **Note: character arrays should have enough space allocated to hold the
                string.  THIS IS THE USER's RESPONSIBILITY ***
c  initialize resolution and layer
                resolution = PGSd_DEM_3ARC
                layer = PGSd_DEM_ELEV
                status = PGS_DEM_GetMetadata(resolution, layer,
     1          pixLatInfo, pixLonInfo, positionUnits, &scaling,
     1          &offset, &fillValue, dataUnits, &mapProjection,
     1          &qualityAssuranceLayer)
                if (status .NE. PGS_S_SUCCESS) then
c  ** Do some error handling
```

NOTES: pixLatInfo and pixLonInfo:

All of the values of this array are in degree decimal format. The first element of the array indicates the spacing between pixels. The second element is the location within the pixel that is used for requesting the location of that pixel (i.e. the center or corner of the pixel). This second element is the vertical (pixLatInfo) or horizontal (pixLonInfo) offset of this location from the top left corner of a pixel.

REQUIREMENTS: PGSTK–0945

ACCESS DEM Quality Data

NAME: **PGS_DEM_GetQualityData()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_GetQualityData(
PGSt_DEM_Tag resolution,
PGSt_integer  qualityField,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
void *qualityData)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getqualitydata(resolution, qualityField, positionCode, latitude, longitude, qualityData)
integer         resolution
integer         qualityField
integer positionCode
double precision        latitude(2)
double precision        longitude(2)
'user defined'  qualityRegion(*)
```

DESCRIPTION: This tool accesses the quality assurance layer of a particular DEM data set. It takes a latitude and longitude of a point of interest and an attribute mask. It returns information concerning the data source, the region over which the quality assurance information is valid, the quality metric of the aforesaid region, or information on the geoid.

INPUTS: resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

qualityField - the type of quality information requested. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the latitude and longitude of the points of interest in decimal format. See Notes to PGS_DEM_SortModels().

OUTPUTS: qualityData - an array containing the quality assurance layer information for the region specified. The information returned is dependent on the flag indicated in the qualityField. For example, one can obtain the data sources for all the data in one's region.

RETURNS: PGS_S_SUCCESS -- success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_SMF_status  status;
PGSt_integer numLayers;
PGSt_integer layerList[1];
PGSt_DEM_Tag resolutionList[1];
PGSt_integer numResolutions;
PGSt_double latitude[2];
PGSt_double longitude[2];
PGSt_integer numVertPix;
PGSt_integer numHorizPix;
PGSt_integer pixByte;
PGSt_integer totalNumPixels;
int16 *qualityData =NULL;
   /*Some initialization.  Initializing resolutions and layers for PGS_DEM
functionality. */
resolutionList[0]= PGSd_DEM_30ARC;
numResolutions = 1;
```

```
layerList[0] = PGSd_DEM_ELEV;
numLayers = 1;
latitude[0] = 40.;
longitude[0] = -100.;
latitude[1] = 38.;
longitude[1] = -97.;
  /*Open the resolution and data layer*/
status = PGS_DEM_Open(resolutionList, numResolutions, layerList,
                                    numLayers);
   if(status != PGS_S_SUCCESS)
   {
         /*ERROR intializing*/
         printf("PGS_DEM_Open: error initializing\n");
   }
   else
   {
         printf("PGS_DEM_Open: Successful Open\n");
   }
   status = PGS_DEM_GetSize(resolutionList[0], PGSd_DEM_GEOID,
                                    PGSd_DEM_DEGREE, latitude, longitude,
                                    &numVertPix, &numHorizPix, &pixByte);
   if(status != PGS_S_SUCCESS)
   {
         /*ERROR with GetSize*/
    printf("PGS_DEM_GetSize: error-- %d\n", status);
   }
   else
   {
         /*print the size of region*/
         printf("PGS_DEM_GetSize: PGSd_DEM_GEOID\n");
```

```
        printf("number of bytes in one pixel is %d\n", pixByte);
        printf("number of pixels vertically spanning region %d\n",
        numVertPix);
        printf("number of pixels horizontally spanning region %d\n",
        numHorizPix);
    }
    /* allocate enough space for qualityData */
    totalNumPixels = numVertPix * numHorizPix;
    qualityData = calloc(totalNumPixels, pixByte);
    if (qualityData == NULL)
    {
        /*error callocing*/
        printf("error callocing\n");
    }
    /* Get Quality Data */
    status = PGS_DEM_GetQualityData(resolutionList[0],
PGSd_DEM_ GEOID, PGSd_DEM_DEGREE, latitude, longitude,
 (void *)qualityData);
    if (status != PGS_S_SUCCESS)
    {
        printf("error: PGS_DEM_GetQualityData\n");
    }
    else
    {
       printf("extracted quality data:using PGS_DEM_GetQualityData\n");
    }
    status = PGS_DEM_Close(resolutionList, numResolutions, layerList,
                numLayers);
    if (status != PGS_S_SUCCESS)
    {
        /*ERROR DE-INITIALIZING*/
```

```
    printf("Error closing DEM session.\n");
    }
```

FORTRAN: TBD

NOTES: All the 15 arc second, 30 arc second, 3 arc second, and 90 arc second DEM data are referenced vertically to mean sea level, which is approximated by the geoid. The numbers for geoid that one can extract using PGS_DEM_GetQualityData, as shown in the example, are added to the DEM value to make the height relative to the WGS84 ellipsoid. Thus in order to get height relative to the WGS84 ellipsoid one calls first PGS_DEM_GetPoint (see example for the function PGS_DEM_GetPoint) to retrieve the elevation data with respect to the mean sea level. The subsequent call to PGS_DEM_GetQualityData, as shown in the example, will retrieve geoid data. Then these values are added together to give the height relative to the WGS84 ellipsoid.

qualityField:

For example, one could query the information on either the data source, the quality metric, or the geoid which corresponds to the flags PGSd_DEM_SOURCE, PGSd_DEM_HORIZONTAL_ACCURACY, PGSd_DEM_VERTICAL_ACCURACY, and PGSd_DEM_GEOID respectively.

qualityData - Followings are the data types and values for quality fields:

Source: Data type is 1 byte integer

code 0-8:

0 - no data (ocean)

1 - Digital Terrain Elevation Data (DTED)

2 - Digital Chart of the World (DCW)

3 - USGS 1-degree DEM's

4 - Army Map Service 1:1,000,000-scale maps

5 - International Map of the World 1:1,000,000-scale map

6 - Peru 1:1,000,000-scale map

7 - New Zealand DEM

8 - Antarctic Digital Database (ADD)

Geoid: Data type is 2 byte integer

Data range is -101 to 75 Meters. Add numbers to Mean Sea Level to achieve WGS84 Geoid.

Method: Data type is 1 byte integer.

accuracy calculation method - code 0-5:

0 - no data (ocean)

1 - accuracy from source DEM metadata

2 - vertical accuracy calculated by comparison with higher resolution DEM; horizontal accuracy from source product specification

3 - accuracy from source DEM product specification

4 - vertical accuracy estimated from contour interval of source;

horizontal accuracy estimated from map scale of source

5 - not calculated

1 is used for DTED.

2 is used for DCW.

3 is used for USGS DEM's.

4 is used for cartographic sources (sources 4, 5, 6, and 7 in source data).

5 is used for Antarctica (where the wide range of contour intervals and map scales in the ADD makes it unreasonable to give a reliable estimate).

Hoizontal Accuracy: Data type is 2 byte integer.

absolute horizontal accuracy: RMSE in meters

-9999 = no data (ocean)

9999 = unknown

Vertical Accuracy: Data type is 2 byte integer.

absolute vertical accuracy: RMSE in meters

-9999 = no data (ocean)

9999 = unknown

REQUIREMENTS: PGSTK–0946

Return Size of Specified DEM Region

NAME: **PGS_DEM_GetSize()**

SYNOPSIS:

C:

```
PGSt_SMF_status
PGS_DEM_GetSize(
PGSt_DEM_Tag  resolution,
PGSt_integer  field,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
PGSt_integer *numPixVertical,
PGSt_integer *numPixHorizontal,
PGSt_integer *sizeDataType)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getsize(resolution, field, positionCode, latitude,
longitude, numPixVertical, numPixHorizontal, sizeDataType)
integer          resolution
integer          field
integer positionCode
double precision      latitude(2)
double precision      longitude(2)
integer numPixVertical
integer numPixHorizontal
integer sizeDataType
```

DESCRIPTION: This tool determines the size of a rectangular region defined by the latitudes and longitudes of its upper left and lower right corners. This tool is meant to facilitate the user's ability to allocate appropriate space for the data returned by PGS_DEM_GetRegion and PGS_DEM_GetQualityData. Use of this tool can prevent core dumps and other errors due to improper allocation of memory.

INPUTS: resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

field - either a mask or a qualityField flag. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the latitude and longitude of the points of interest. See Notes to PGS_DEM_SortModels().

OUTPUTS: numPixVertical - a pointer to the number of pixels spanning the vertical extent of the region

numPixHorizontal - a pointer to the number of pixels spanning the horizontal extent of the region

sizeDataType - a pointer to the size of an individual pixel of data, in bytes

RETURNS: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

EXAMPLES:

C:

```
PGSt_SMF_status status;

PGSt_integer resolution;

PGSt_integer layer;

PGSt_double latitude[2];

PGSt_double longitude[2];

PGSt_integer numVertPix;

PGSt_integer numHorizPix;

PGSt_integer pixByte;

/* initialize resolution and layer */
```

```
                resolution = PGSd_DEM_30ARC;

                layer = PGSd_DEM_ELEV;

/*initialize location of region.  In this case, position is in signed decimal
                degrees */

                latitude[0] = 4.0;

                longitude[0] = 112.0;

                latitude[1] =  -3.0;

                longitude[1] = 115.5;

                status = PGS_DEM_GetSize(resolution, layer, PGSd_DEM_DEGREE,
                latitude, longitude, &numVertPix, &numHorizPix, &pixByte);

                if(status != PGS_S_SUCCESS)

                {

/* Do some error handling ...*/

                ....
```

FORTRAN:

```
                integer resolution

                integer layer

                integer status

                double precision latitude(2)

                double precision longitude(2)

                integer numVertPix

                integer numHorizPix

                integer pixByte


C   **initialize resolution and layer

                resolution = PGSd_DEM_30ARC

                layer = PGSd_DEM_ELEV

C   **initialize location of region.  In this case, position is in signed
                decimal degrees

                latitude(1) = 4.0
```

```
                longitude(1) = 112.0

                latitude(2) =  -3.0

                longitude(2) = 115.5

                status = PGS_DEM_GetSize(resolution, layer,

     1          PGSd_DEM_DEGREE, latitude, longitude, numVertPix,

     1           numHorizPix, pixByte)

                if(status .NE. PGS_S_SUCCESS) then

C  ** Do some error handling ...**
```

NOTES: field:

This indicates the layer attribute or field of the quality assurance layer over which the region is "sized". For ECS Deliveries B.0, the layers that may be inputted are elevation, standard deviation of elevation, water/land, slope gradient, standard deviation of slope gradient, aspect , data source, quality metric, and geoid which correspond to the flags PGSd_DEM_ELEV, PGSd_DEM_STDEV_ELEV, PGSd_DEM_WATER_LAND, PGSd_DEM_SLOPE, PGSd_DEM_STDEV_SLOPE, PGSd_DEM_ASPECT, PGSd_DEM_SOURCE PGSd_DEM__HORIZONTAL_ACCURACY, PGSd_DEM_VERTICAL_ACCURACY, and PGSd_DEM_GEOID, respectively. The layers that will be available in the future are: topographical obscuration (PGSd_DEM_TOP_OBSC), and topographical shadow (PGS_DEM_TOP_SHAD). Note that for 90 arc second data the only available layer are elevation and land/water. And for 15 arc second data the only available layer are elevation, land/water and standard deviation of Elevation.

REQUIREMENTS: PGSTK–0947

6.3.2 Ancillary Data Tools

6.3.2.1 Introduction

There will be a large number of ancillary data files used in ECS instrument processing. The tools in this section address files already identified at this writing.

Users could utilize language standard input/output functions or the HDF tools to access the ancillary data. However, a suite of higher level tools is required for the following reasons:

a. to enable data from locations specified by the user to be returned to the user thus avoiding having to know the internal structure of the file.

b. to shield the user from having to know details of parameter source or source format or to track changes in either, although source changes will be agreed upon with the user.

c. to provide for certain additional manipulations of extracted data.

For this final point (c), only those data sets that have been specifically identified as requiring particular manipulations will be serviced; i.e., the ancillary tools do not intend to provide a general manipulation service for all types of data. However, the tools that extract from location (a) will be sufficiently generic to allow additional data sets of a similar type to be used.

NAME: **PGS_AA_dcw( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_Status
PGS_AA_dcw (char iparms[][100],     coverage name—PO
        PGSt_integer nParms,        number of coverages
        PGSt_double longitude[],    longitude of point(s)
        PGSt_double latitude[],     latitude of point(s)
        PGSt_integer npoints,       number of points
        void *results)              result of search
```

FORTRAN:

```
include'PGS_AA_10.f'

integer function
PGS_AA_dcw(parms, nParms, latitude, longitude, npoints, results)
        character*99  iparms(*),
        integer       nParms,
        double        latitude(*)
```

DESCRIPTION: This routine receives either a single point or an array of location points and navigates the DCW database in order to find the coverage that the user supplies as parm. Once the coverage is identified, the database path is updated, with each file and table identified, until the table containing the locational information is located. Once this table is found, the table is opened and the result for a latitude/longitude is extracted and returned in results.

```
[start]
PERFORM    PGS_AA_dcw_Parm
PERFORM    PGS_AA_dcw_Intile
PERFORM    PGS_AA_dcw_Inface
PERFORM    PGS_AA_dcw_Feature
PERFORM    return PGS_S_SUCCESS

[end]
```

INPUTS:

Table 6-138. PGS_AA_dcw Inputs

Name	Description	Units	Min	Max
parms	parameter wanted	N/A	N/A	N/A
nParms	number of parameters	N/A	1	1
latitude	latitude location	degrees	-90.0	90.0
longitude	longitude location	degrees	-180.0	180.0
npoints	number of points	N/A	0	Unlimited

OUTPUTS:

Table 6-139. PGS_AA_dcw Outputs

Name	Description	Units	Min	Max
results	extracted parameter	char	N/A	N/A

RETURNS:

Table 6-140. PGS_AA_dcw Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSAA_E_DCW_ERROR	Error in extracting value required
PGSAA_W_DCW_NODATA	No data at that point in data base

The following errors are reported to the error log

PGSAA_E_CANT_FIND_PARM
PGSAA_E_CANT_GET_CONTINENT_PATH
PGSAA_E_CANT_GET_TILE_DIR
PGSAA_E_CANT_GET_POINT_IN_FACE
PGSAA_E_CANT_GET_POINT_INFO

EXAMPLES:

C:

```
#include <PGS_AA.h>

PGSt_double latitude[2] = {-9.29, -25.34};
PGSt_double longitude[2] = {110.3, 30.9};
PGSt_integer results[2];
char parm[PGSd_AA_MAXNOCACHES][100] = {"po"};

ret_status = PGS_AA_dcw(parm, 1, longitude, latitude, 2,
                        results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           PGS_AA_dcw
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms(2)
double            latitude(2)
double            longitude(2)
integer           npoints(2)
integer           result(2)
parms(1)= "po"
nParms = 2
latitude = -9.29, -25.34
longitude = 110.3, 30.9
npoints = 2

call pgs_aa_dcw(parms, nParms, longitude, latitude, npoints,
                results)
```

NOTES: For further details of the background to this tool see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide).

IMPORTANT: The PGS_AA_dcw code calls a number of library modules, which carry out such actions as mallocing memory for files, opening files, opening tables, reading tables, extracting information from tables and closing tables. These library modules are detailed in the DCW format specification and the associated vector product format (VPF) library software.

NOTE: Precision of latitude and longitude is machine specific, not data–base specific.

REQUIREMENTS: PGSTK–0840, PGSTK–0870, PGSTK–1360, PGSTK–1362

Access Available Data from a Set of Standard Digital Elevation Models (DEMs)

NAME: **PGS_AA_dem( )**

SYNOPSIS:

C:

```
#include "PGS_AA.h"

PGSt_SMF_status
PGS_AA_dem(char parms[][100],
        PGSt_integer        nParms,
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_integer        versionFlag[],
        PGSt_integer        nPoints,
        PGSt_PC_Logical     fileId,
        PGSt_integer        operation,
        void                *results)
```

FORTRAN:

```
include"PGS_AA_10.f"
include"PGS_AA.f"

integer function
pgs_aa_dem (parms, nparms, latitude, longitude,
                versionflag, npoints, fileId, operation, results)
        character*99        parms(*)
        integer             nParms
        double precision    latitude(*)
        double precision    longitude(*)
        integer             versionflag(*)
        integer             npoints
        integer             fileId
        integer             operation
        'user specified'    results (see Notes)
```

DESCRIPTION: This routine provides the interface to retrieve DEM values from the gridded data set.

INPUTS:

Table 6-141. PGS_AA_dem Inputs

Name	Description	Units	Min	Max
parms	parameter names requested	see notes		
nParms	number of parms	none	1	#defined
latitude	latitude(s) of the requested point	degrees	-90.00	90.00
longitude	longitude(s) of the requested point	degrees	-180.00	180.00
nPoints	no. of points requested	none	1	variable
fileId	logical file number	none	variable	variable
operation	defines user required	none	1	variable

OUTPUTS:

Table 6-142. PGS_AA_dem Outputs

Name	Description	Units	Min	Max
versionFlag	indicates tile location for a point (see notes)	see notes	1	variable
results	results	see notes		

RETURNS:

Table 6-143. PGS_AA_dem Returns

Returns	Descriptions
PGS_S_SUCCESS	Successful return
PGSAA_E_NPOINTSINVALID	Number of points invalid
PGSAA_E_TILE_STATUS	Could not establish tile status of the DEM file
PGSAA_E_2DGEO	Error returned from PGS_AA_2Dgeo
PGSAA_E_SUPPORTID	Could not establish support file id
PGSAA_E_MINMAX	Could not establish min/max range for the DEM
PGSAA_E_DATATYPE	Could not establish parameter datatype
PGSAA_E_UNKNOWN_DATATYPE	DEM datafile datatype is unknown

EXAMPLES:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] = { "USAelevation" };
long nParms = 1;
PGSt_double latitude[MAX_POINTS] = {51.5, 51.23666,
                                    50.973333} ;
PGSt_double longitude[MAX_POINTS] = {0.1666666,0.3832,
                                     0.5999};

PGSt_integer  versionFlag[MAX_POINTS];

PGSt_integer nPoints = 3;
long fileId = 210;
long version = 0;
long operation = 1;
short results[3];
retStatus = PGS_AA_2Dgeo(parms, nParms, latitude, longitude,
                         versionFlag, nPoints, fileId,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

parms(PGSd_AA_MAXNOCACHES)
integer                 pgs_aa_dem
integer                 versionFlag(300)
integer                 nParm
double precision        latitude(300)
double precision        longitude(300)
integer                 fileId
integer                 nPoints
integer                 version
integer                 operation
integer                 results(300)
integer                 retStatus
parms(1)= "USAelevation"
nParms = 1
fileId = 202
operation = 1
nPoints = 300
do 10 i = 1, 300
.
```

```
                .
                latitude(i) = calculated_user_lat
                longitude(i) = calculated_user_lon
                .
                .
 10    continue

                retStatus = pgs_aa_dem( parms, nparms, latitude,
                2           longitude, versionFlag, npoints,
                1             fileId, operation, results )
```

NOTES:

The added facility that differentiates this tool from its sister tool PGS_AA_2Dgeo is that this routine can handle tiled data sets by selecting from geographically separated tiles. Some of the DEM datafiles can be very large files and are necessarily tiled into smaller files to avoid memory problems.

Also this routine processes all input point data and returns a warning if some of the input points were found to be out of range. In such an event user can examine versionFlag[] to locate the offending points. For such points the corresponding location in versionFlag would contain a value PGSd_AA_OUT_OF_RANGE, e.g.,

if latitude[3] and longitude[3] is the offending point then

versionFlag[3] = PGSd_AA_OUT_OF_RANGE.

For other points the versionFlag[] would actually contain the number of the tile where the point was located.

For the details of DEM datafiles the user is referred to appendix D.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The results buffer holds the final output sent back to the user. It can hold data of 4 types (long, short, float, double).

For more details the user is referred to information regarding PGS_AA_2Dgeo.

Users MUST be aware of the amount of disk space required by the number of calls to the tool (where calls demand the ingestion of separate physical files), and in doing so not exceed the capacity of the machine they are working on.

DEC—users

DEC users should be aware that for some of the product files a DEC version (e.g., etop05.dat_dec) is supplied. The user should use these instead of the normal files. This is for backward compatibility with the PGS_AA_2Dgeo tool. For the rest of the data files there is an inbuilt facility to swap the bytes. For these files there is a flag 'swapBytes = yes' in the support file. This flag is set to 'no' for the data files with 'dec' versions.

Another issue that the user should be aware of is that DEC represents 'long' datatype as 8 bytes long. Therefore, if there is a datafile created on a different platform (most other platforms represent 'long' as 4 bytes), then that file must be converted first to be used on the DEC. Conversion should simply be reading the file as 'int' (4 bytes) and writing it out as 'long' (8 bytes) on the DEC. To take care of byteswapping the support file for such datafile should contain a flag 'swapBytes = yes'.

REQUIREMENTS: PGSTK–0840, PGSTK–0980

Extract String Parameter from Parameter=Value Formatted File

NAME: **PGS_AA_PeVA_string( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_string(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        char                *value[] )
```

FORTRAN:

```
include"PGS_AA_10.f"
include"PGS_AA.f"

integer function
pgs_aa_peva_string( pevLogical, parameter, value )
        integer         pevLogical
        character*(*)   parameter
        character*(*)   value
```

DESCRIPTION: This routine returns the value associated with a string type parameter from the given file.

INPUTS:

Table 6-144. PGS_AA_PeVA_string Inputs

Name	Description	Units	Min	Max
pevLogical	file logical for file to be accessed	see notes		
parameter	name of parameter to be retrieved	see notes		

OUTPUTS:

Table 6-145. PGS_AA_PeVA_string Outputs

Name	Description	Units	Min	Max
value	value associated with retrieved parameter	see notes		

RETURNS:

Table 6-146. PGS_AA_PeVA_string Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSAA_E_PEV_ERROR	Error in extracting the required value

The following errors are reported to the error log

PGSAA_E_CANT_GET_FILE_ID
PGSAA_E_CANT_OPEN_INPUT_FILE
PGSAA_E_AGG_CANT_BE_INSERTED
PGSAA_E_READLABEL_PARSE_ERROR
PGSAA_E_PARAMETER_INVALID
PGSAA_E_FIRST_NODE_NOT_FOUND

EXAMPLE:

C:

```
#include    <PGS_AA.h>
#define     MAX_STRING 30

PGSt_SMF_status retStatus;
char        *myStringValue[MAX_STRING]= {"          "
                                         "          "
                                         "
"};

ret_status = PGS_AA_PeVA_string(MY_PEV_FILE,
                                "MY_STRING_PARAMETER",
                                 myStringValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_string
integer           pevLogical, return
character*30      parameter
character*20      value
```

```
pevLogical = 876
parameter = "dataType"

return = pgs_aa_peva_string( pevLogical, parameter, value )
```

NOTES: The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is also a string as returned from the data file identified by the logical. For

REQUIREMENTS: PGSTK–1365

Extract Real Parameter from Parameter = Value Formatted File

NAME: **PGS_AA_PeVA_real( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_real(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        PGSt_double         *value )
```

FORTRAN:

```
include'PGS_AA_10.f'
include'PGS_AA.f'

integer function
pgs_aa_peva_real( pevLogical, parameter, value )
        integer             pevLogical
        character*(*)       parameter
        double precision    value
```

DESCRIPTION: This routine returns the value associated with a string type parameter from the given file.

INPUTS:

Table 6-147. PGS_AA_PeVA_real Inputs

Name	Description	Units	Min	Max
pevLogical	file logical for file to be accessed	see notes		
parameter	name of parameter to be retrieved	see notes		

OUTPUTS:

Table 6-148. PGS_AA_PeVA_real Outputs

Name	Description	Units	Min	Max
value	value associated with retrieved parameter	see notes		

RETURNS:

Table 6-149. PGS_AA_PeVA_real Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSAA_E_PEV_ERROR	Error in extracting the required value

The following errors are reported to the error log

PGSCUC_E_CANT_GET_FILE_ID
PGSCUC_E_CANT_OPEN_INPUT_FILE
PGSCUC_E_AGG_CANT_BE_INSERTED
PGSCUC_E_READLABEL_PARSE_ERROR
PGSCUC_E_PARAMETER_INVALID
PGSCUC_E_FIRST_NODE_NOT_FOUND

EXAMPLE:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status   retStatus;
PGSt_double       myRealValue[10];

ret_status = PGS_AA_PeVA_real(MY_PEV_FILE,
                              "MY_STRING_PARAMETER",
                               &myRealValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_real
integer           pevLogical, return
character*30      parameter
double precision  value
pevLogical = 876
parameter = "maxLat"

return = pgs_aa_peva_real( pevLogical, parameter, value )
```

NOTES: The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is a real as returned from the data file identified by the logical.

REQUIREMENTS: PGSTK–1365

Extract Integer Parameter from Parameter = Value Formatted File

NAME: **PGS_AA_PeVA_integer( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_integer(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        PGSt)integer        *value)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_peva_integer( pevLogical, parameter, value )
        integer         pevLogical
        character*(*)   parameter
        integer         value
```

DESCRIPTION: This routine returns the value associated with a string type parameter from the given file.

INPUTS:

Table 6-150. PGS_AA_PeVA_integer Inputs

Name	Description	Units	Min	Max
pevLogical	file logical for file to be accessed	see notes		
parameter	name of parameter to be retrieved	see notes		

OUTPUTS:

Table 6-151. PGS_AA_PeVA_integer Outputs

Name	Description	Units	Min	Max
value	value associated with retrieved parameter	see notes		

RETURNS:

Table 6-152. PGS_AA_PeVA_integer Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSAA_E_PEV_ERROR	Error in extracting the required value

The following errors are reported to the error log

PGSCUC_E_CANT_GET_FILE_ID
PGSCUC_E_CANT_OPEN_INPUT_FILE
PGSCUC_E_AGG_CANT_BE_INSERTED
PGSCUC_E_READLABEL_PARSE_ERROR
PGSCUC_E_PARAMETER_INVALID
PGSCUC_E_FIRST_NODE_NOT_FOUND

EXAMPLE:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status   retStatus;
PGSt_integer      myIntValue;

ret_status = PGS_AA_PeVA_integer(MY_PEV_FILE,
                                 "MY_STRING_PARAMETER",
                                  &myIntValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_integer
integer           pevLogical, return
character*30      parameter
integer*(*)       value
pevLogical = 876
parameter = "size"

return = pgs_aa_peva_real( pevLogical, parameter, value )
```

NOTES: The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is a real as returned from the data file identified by the logical.

REQUIREMENTS: PGSTK–1365

Extract Data from Gridded Data Sets by Geographic Location

NAME: **PGS_AA_2Dgeo( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_2Dgeo ( char  iparms[][100],
        PGSt_integer  nParms,
        PGSt_double  latitude[],
        PGSt_double  longitude[],
        PGSt_integer  nPoints,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void          *results);
```

FORTRAN:

```
include'PGS_AA_10.f'
include'PGS_AA.f'

integer function
pgs_aa_2dgeo( parms, nparms, latitude, longitude, fileId,
                version, operation, results )
        character*99        parms(*)
        integer             nParms
        real*8              latitude(*)
        real*8              longitude(*)
        integer             fileId
        integer             version
        integer             operation
        'user specified'    results (see Notes)
```

DESCRIPTION: The user specifies a parameter, a fileId and version from which the data are to be extracted using the geographic coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that extracts gridded data by geographic location.

```
[start]
PERFORM    PGS_AA_Map
PERFORM    PGS_AA_GetSupp     to get support data
DO         allocate memory to parmBuffer using
```

totalParmMemoryCache
PERFORM PGS_AA_FF_Setup
DO set tool used to 2
PERFORM PGS_AA_GEOGrid
[end]

INPUTS:

Table 6-153. PGS_AA_2Dgeo Inputs

Name	Description	Units	Min	Max
parms	parameter names	see notes	requested	
nParms	number of parms	none	1	#defined
latitude	latitude(s) of the requested point	degrees	-90.00	90.00
longitude	longitude(s) of the requested point	degrees	-180.00	180.00
nPoints	no. of points requested	none	1	variable
fileId	logical file number	none	variable	variable
version	version of dynamic file	none	1	variable
operation	defines user required	none	1	variable

OUTPUTS:

Table 6-154. PGS_AA_2Dgeo Outputs

Name	Description	Units	Min	Max
results	results	see notes		

RETURNS:

Table 6-155. PGS_AA_2Dgeo Returns

To User/Log File	Return	Description
u	PGSAA_E_GEOERRO	Error in GEO extraction
l	PGSAA_E_AUTOOPERATION	Error in executing autoOperation
l	PGSAA_E_AUTOOPERATIONUNSET	No autoOperation found in support file
l	PGSAA_E_OPERATION	Error in executing operation
l	PGSAA_E_OPERATIONUNSET	Operation not set by user
ul	PGSAA_E_GEOTOSTRUCT	Failure in calculation of structure from lat/lon
ul	PGSAA_E_UNIDENTIFIEDTYPE	Type cannot be identified, results failure
u	PGSAA_E_SUPPORTFILE	Support or format files inaccessible
ul	PGSAA_E_PARMSNOTFOUND	Parameter(s) not found in the support support file
l	PGSAA_E_DATARATEUNSET	dataRate attribute unset in support file
ul	PGSAA_E_PARMSFROMANYFILES	Parameters requested from more than one physical file
ul	PGSAA_E_INVALIDNOPARMS	No of parms incorrect
ul	PGSAA_E_BADSUPPSUPPORT	Tool support file is corrupted or incomplete
ul	PGSAA_E_CANTFINDFILE	Format of input data file inaccessible
u	PGSAA_E_FFERROR	A freeform error has occured
l	PGSAA_E_FFDBIN	Failure in Freeform make_dbin function
l	PGSAA_E_FFDBSET	Failure in Freeform db_set function
l	PGSAA_E_FFDBEVENTS	Failure in Freeform db_events function
ul	PGSAA_E_MALLOC	Failure to malloc
u	PGSAA_E_PEV_ERROR	An error has occurred in the PeVA tool
l	PGSAA_E_PEV_XS_SUPPFILES	Too many PeVA files open, increase MAXFILES
l	PGSAA_E_CANT_GET_VALUE	Unable to extract value from dbin
l	PGSAA_E_GETDBIN	Error in PeVA tool obtaining dbin
u	PGSAA_E_GETSUPP	An error was detected while extracting support data
l	PGSAA_E_POSITION_CALC_FAILURE	The position in the parmBuffer of the requested values was miscalculated
u	PGSAA_E_TWOD_READ_ERROR	Function failure to read parameter values from buffer
l	PGSAA_E_EXTRACTORESULTSERROR	Failure to transfer selected values from parmBuffer to results
ul	PGSAA_E_OUTOFRANGE	Input values out of data set range

EXAMPLE:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] = {
                                        "etop05SeaLevelElevM" };
long nParms = 1;

PGSt_double latitude[] = {51.5, 51.23666, 50.973333} ;
PGSt_double longitude[] = {0.1666666,0.3832,  0.5999};

PGSt_integer nPoints = 3;

PGSt_integer fileId = 10955;

PGSt_integer version = 1;

PGSt_integer operation = 1;

short results[3];

retStatus = PGS_AA_2Dgeo(parms, nParms, latitude, longitude,
                         nPoints, fileId, version,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_2dgeo
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
real*8            latitude(300)
real*8            longitude(300)
integer           fileId
integer           nPoints
integer           version
integer           operation
integer           results(300)
parms(1)= "fnocMod"
nParms = 1
fileId = 10965
operation = 1
version = 1
nPoints = 300
do 10 i = 1, 300
.
.
```

```
            latitude(i) = calculated_user_lat
            longitude(i) = calculated_user_lon
            .
            .

10          continue

            call pgs_aa_2dgeo( parms, nparms, latitude, longitude,
                               fileId, version, operation, results )
```

NOTES: For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The parmBuffer input is a memory buffer holding whatever data is extracted form the data set requested by the user. The results buffer is similar although holds the final output sent back to the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

REQUIREMENTS: PGSTK–0840, PGSTK–0931, PGSTK–0980, PGSTK–1030, PGSTK–1362

Extract Data from Gridded Data Sets by Geographic Location

NAME: **PGS_AA_3Dgeo( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_3Dgeo ( char iparms[][100],
        PGSt_integer  nParms,
        PGSt_double   latitude[],
        PGSt_double   longitude[],
        PGSt_integer  height[],
        PGSt_integer  nPoints,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void          *results);
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

        character*99        iparms(PGSd_AA_MAXNOCACHES)
        integer             nParms
        real*8              latitude(*)
        real*8              longitude(*)
        integer             height(*)
        integer             fileId
        integer             version
        integer             operation
        'user specified'    results (see Notes)

integer function
pgs_aa_3dread( parms, nparms, latitude, longitude,
                height, fileId, version, operation, results )
```

DESCRIPTION: The user specifies a parameter, a fileId and version from which the data are to be extracted using the geographic coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that extract gridded data by geographic location.

```
[start]
PERFORM    PGS_AA_Map
PERFORM    PGS_AA_GetSupp     to get support data
```

```
DO          allocate memory to parmBuffer using
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
DO          set tool used to 3
PERFORM     PGS_AA_GEOGrid
[end]
```

INPUTS:

Table 6-156. PGS_AA_3Dgeo Inputs

Name	Description	Units	Min	Max
iparms	parameter names requested	see notes		
nParms	number of parms	none	1	4
latitude	latitude(s) of the requested point	degrees	-90.00	90.00
longitude	longitude(s) of the requested point	degrees	-180.00	180.00
height	height of the requested point	none	1	variable
nPoints	no. of points requested	none	1	variable
fileId	logical file number	none	variable	variable
version	version of dynamic file	none	1	variable
operation	defines user required	none	1	variable

OUTPUTS:

Table 6-157. PGS_AA_3Dgeo Outputs

Name	Description	Units	Min	Max
results	results	see notes		

RETURNS:

Table 6-158. PGS_AA_3Dgeo Returns (1 of 2)

To User/Log File	Return	Description
u	PGSAA_E_GEOERROR	Error in GEO extraction
l	PGSAA_E_AUTOOPERATION	Error in executing autoOperation
l	PGSAA_E_AUTOOPERATIONUNSET	No autoOperation found in support file
l	PGSAA_E_OPERATION	Error in executing operation
l	PGSAA_E_OPERATIONUNSET	Operation not set by user
ul	PGSAA_E_GEOTOSTRUCT	Failure in calculation of structure from lat/lon
ul	PGSAA_E_UNIDENTIFIEDTYPE	Type cannot be identified, results failure
u	PGSAA_E_SUPPORTFILE	Support or format files inaccessible

Table 6-158. PGS_AA_3Dgeo Returns (2 of 2)

To User/Log File	Return	Description
ul	PGSAA_E_PARMSNOTFOUND	Parameter(s) not found in the support support file
l	PGSAA_E_DATARATEUNSET	dataRate attribute unset in support file
ul	PGSAA_E_PARMSFROMANYFILES	Parameters requested from more than one physical file
ul	PGSAA_E_INVALIDNOPARMS	No of parms incorrect
ul	PGSAA_E_BADSUPPSUPPORT	Tool support file is corrupted or incomplete
ul	PGSAA_E_CANTFINDFILE	Format of input data file inaccessible
u	PGSAA_E_FFERROR	A freeform error has occured
l	PGSAA_E_FFDBIN	Failure in Freeform make_dbin function
l	PGSAA_E_FFDBSET	Failure in Freeform db_set function
l	PGSAA_E_FFDBEVENTS	Failure in Freeform db_events function
ul	PGSAA_E_MALLOC	Failure to malloc
u	PGSAA_E_PEV_ERROR	An error has occurred in the PeVA tool
l	PGSAA_E_PEV_XS_SUPPFILES	Too many PeVA files open, increase MAXFILES
l	PGSAA_E_CANT_GET_VALUE	Unable to extract value from dbin
l	PGSAA_E_GETDBIN	Error in PeVA tool obtaining dbin
u	PGSAA_E_GETSUPP	An error was detected while extracting support data
l	PGSAA_E_POSITION_CALC_FAILURE	The position in the parmBuffer of the requested values was miscalculated
u	PGSAA_E_THREED_READ_ERROR	Function failure to read parameter values from buffer
l	PGSAA_E_EXTRACTORESULTSERROR	Failure to transfer selected values from parmBuffer to results
ul	PGSAA_E_OUTOFRANGE	Input values out of data set range

EXAMPLE:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] =
                          {"nmcRucSigPres","nmcRucSigPot"};
long nParms = 2;

PGSt_double latitude[] = {51.5, 51.23666, 50.973333} ;
PGSt_double longitude[] = {0.1666666, 0.3832, 0.5999};
long height[] = {1,2,1};
PGSt_integer nPoints = 3;

long fileId = 10972;

long version = 1;
```

```
long operation = 2;

short results[3][2];

retStatus = PGS_AA_3Dgeo(parms, nParms, latitude, longitude,
                         height, nPoints, fileId, version,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_3dgeo
character*99      iparms(PGSd_AA_MAXNOCACHES)
integer           nParms
real*8            latitude(300)
real*8            longitude(300)
integer           height(300)
integer           fileId
integer           nPoints
integer           version
integer           operation
integer           results(2,300)
parms(1)= "nmcRucSigPres"

parms(2)= "nmcRucSigPot"
nParms= 2
fileId = 10972
operation = 2
version = 1
nPoints = 300
do 10 i = 1, 300
.
.
latitude(i) = calculated_user_lat
longitude(i) = calculated_user_lon
height(i) = calculate_user_height
.
.

10      continue

call pgs_aa_3dgeo( iparms, nparms,latitude, longitude,
                   height, fileId, version, operation,
                   results )
```

NOTES: For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

Height or the z dimension is a layer number in the file and is data set dependent.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

REQUIREMENTS: PGSTK–0931, PGSTK–0840, PGSTK–1362

Extract Data from Gridded Data Sets by File Structure

NAME: **PGS_AA_2DRead( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_2DRead(
        char iparms[][100],
        PGSt_integer  nParms,
        PGSt_integer  xStart,
        PGSt_integer  yStart,
        PGSt_integer  xDim,
        PGSt_integer  yDim,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void          *results)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_2dread( parms, nparms, xStart, yStart, xDim, yDim, fileId,
                version, operation, results )
        character*99        parms(*)
        integer             nParms,
        integer             xStart,
        integer             yStart,
        integer             xDim,
        integer             yDim,
        integer             fileId,
        integer             version,
        integer             operation,
        'user specified'    results (see Notes)
```

DESCRIPTION: The user specifies a parameter, a fileId and version from which the data are to be extracted using the data structure coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that accepts the arguments and calls PGS_AA_Map, PGS_AA_GetSupp, PGS_AA_FF_Setup and

PGS_AA_2DReadGrid. The first 3 of these modules determine the validity of the call and initialize support and load the identified data into memory. PGS_AA_2DReadGrid performs the extraction requested from the input arguments

```
[start]
PERFORM     PGS_AA_Map
PERFORM     PGS_AA_GetSupp     to get support data
DO          allocate memory to parmBuffer using
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
PERFORM     PGS_AA_2DReadGrid
[end]
```

INPUTS:

Table 6-159. PGS_AA_2DRead Input

Name	Description	Units	Min	Max
parms	parameter names requested	see notes		
nParms	number of parms	none	1	#defined
xStart	the x start point	none	1	variable
yStart	the y start point	none	1	variable
xDim	the x dimension	none	1	variable
yDim	the y dimension	none	1	variable
fileId	logical file number	none	variable	variable
version	version of dynamic file	none	1	variable
operation	defines user required	none	1	variable

OUTPUTS:

Table 6-160. PGS_AA_2DRead Output

Name	Description	Units	Min	Max
results	results	variable	N/A	N/A

RETURNS:

Table 6-161. PGS_AA_2DRead Returns

To User/Log File	Return	Description
u	PGSAA_E_SUPPORTFILE	Support or format files inaccessible
ul	PGSAA_E_PARMSNOTFOUND	Parameter(s) not found in the support support file
l	PGSAA_E_DATARATEUNSET	dataRate attribute unset in support file
ul	PGSAA_E_PARMSFROMANYFILES	Parameters requested from more than one physical file
ul	PGSAA_E_INVALIDNOPARMS	No of parms incorrect
ul	PGSAA_E_BADSUPPSUPPORT	Tool support file is corrupted or incomplete
ul	PGSAA_E_CANTFINDFILE	Format of input data file inaccessible
u	PGSAA_E_FFERROR	A freeform error has occured
l	PGSAA_E_FFDBIN	Failure in Freeform make_dbin function
l	PGSAA_E_FFDBSET	Failure in Freeform db_set function
l	PGSAA_E_FFDBEVENTS	Failure in Freeform db_events function
ul	PGSAA_E_MALLOC	Failure to malloc
u	PGSAA_E_PEV_ERROR	An error has occurred in the PeVA tool
l	PGSAA_E_PEV_XS_SUPPFILES	Too many PeVA files open, increase MAXFILES
l	PGSAA_E_CANT_GET_VALUE	Unable to extract value from dbin
l	PGSAA_E_GETDBIN	Error in PeVA tool obtaining dbin
u	PGSAA_E_GETSUPP	An error was detected while extracting support data
l	PGSAA_E_POSITION_CALC_FAILURE	The position in the parmBuffer of the requested values was miscalculated
u	PGSAA_E_TWOD_READ_ERROR	Function failure to read parameter values from buffer
l	PGSAA_E_EXTRACTORESULTSERROR	Failure to transfer selected values from parmBuffer to results
ul	PGSAA_E_OUTOFRANGE	Input values out of data set range

EXAMPLE:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

short results[50][20]
char parm[PGSd_AA_MAXNOCACHES][100] =
                                {"OlsonWorldEcosystems1.3a"};
PGSt_integer      nParms = 1;
PGSt_integer      xStart = 4;
PGSt_integer      yStart = 7;
PGSt_integer      xDim = 20;
```

```
PGSt_integer      yDim = 50;
PGSt_integer      fileId = 10952;

PGSt_integer version = 1;

PGSt_SMF_status = PGS_AA_2DRead (parm, nParms, xStart,
                                 yStart, xDim, yDim, fileId,
                                 version, 1, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_2dread
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
integer           xStart
integer           yStart
integer           xDim
integer           yDim
integer           fileId
integer           version
integer           operation
PGSt_integer      results(14, 20)
parms(1)= "OlsonWorldEcosystems1.3a"
nParms = 1
yStart = 102
xStart = 205
yDim = 20
xDim = 14
fileId = 10952
operation = 1
version = 1

call pgs_aa_2dread( parms, nparms, xStart, yStart, xDim,
                    yDim, fileId, version, operation,
                    results )
```

NOTES: For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

REQUIREMENTS: PGSTK–0931, PGSTK–0980, PGSTK–1000, PGSTK–1030, PGSTK–1360, PGSTK–1362

Extract Data from Gridded Data Sets by File Structure Parameters

NAME: **PGS_AA_3DRead( )**

SYNOPSIS:

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_3DRead(
        char iparms[][100],
        PGSt_integer nParms
        PGSt_integer xStart
        PGSt_integer yStart
        PGSt_integer zStart
        PGSt_integer xDim
        PGSt_integer yDim
        PGSt_integer zDim
        PGSt_integer fileId
        PGSt_integer version
        PGSt_integer operation
        void *results)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_3dread( iparms, nparms, xStart, yStart, zStart, xDim, yDim, zDim,
                fileId, version, operation, results )
        character*99        iparms(*)
        integer         nParms,
        integer         xStart,
        integer         yStart,
        integer         zStart,
        integer         xDim,
        integer         yDim,
        integer         zDim,
        integer         fileId,
        integer         version,
        integer         operation,
        'user specified' results (see Notes)
```

DESCRIPTION: The user specifies a parameter, a fileId and version from which the data are to be extracted using the file structure coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that accepts the arguments and calls PGS_AA_Map, PGS_AA_GetSupp, PGS_AA_FF_Setup and PGS_AA_3DReadGrid. The first 3 of these modules determine the validity of the call and initialize support and load the identified data into memory. PGS_AA_3DReadGrid performs the extraction requested from the input arguments

```
[start]
PERFORM     PGS_AA_Map
PERFORM     PGS_AA_GetSupp     to get support data
DO          allocate memory to parmBuffer using
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
PERFORM     PGS_AA_3DReadGrid
[end]
```

INPUTS:

Table 6-162. PGS_AA_3DRead Inputs

Name	Description	Units	Min	Max
parms	parameter names requested	see notes		
nParms	number of parms	none	1	#defined
xStart	the x start point	none	1	variable
yStart	the y start point	none	1	variable
zStart	he z start point	none	1	variable
xDim	the x dimension	none	1	variable
yDim	the y dimension	none	1	variable
zDim	the z dimension	none	1	variable
fileId	logical file number	none	variable	variable
version	version of dynamic file	none	1	variable
operation	defines user required	none	1	variable

OUTPUTS:

Table 6-163. PGS_AA_3DRead Outputs

Name	Description	Units	Min	Max
results	results	see notes		

RETURNS:

Table 6-164. PGS_AA_3DRead Returns

To User/Log File	Return	Description
u	PGSAA_E_SUPPORTFILE	Support or format files inaccessible
ul	PGSAA_E_PARMSNOTFOUND	Parameter(s) not found in the support support file
l	PGSAA_E_DATARATEUNSET	dataRate attribute unset in support file
ul	PGSAA_E_PARMSFROMANYFILES	Parameters requested from more than one physical file
ul	PGSAA_E_INVALIDNOPARMS	No of parms incorrect
ul	PGSAA_E_BADSUPPSUPPORT	Tool support file is corrupted or incomplete
ul	PGSAA_E_CANTFINDFILE	Format of input data file inaccessible
u	PGSAA_E_FFERROR	A freeform error has occured
l	PGSAA_E_FFDBIN	Failure in Freeform make_dbin function
l	PGSAA_E_FFDBSET	Failure in Freeform db_set function
l	PGSAA_E_FFDBEVENTS	Failure in Freeform db_events function
ul	PGSAA_E_MALLOC	Failure to malloc
u	PGSAA_E_PEV_ERROR	An error has occurred in the PeV tool
l	PGSAA_E_PEV_XS_SUPPFILES	Too many PeVA files open, increase MAXFILES
l	PGSAA_E_CANT_GET_VALUE	Unable to extract value from dbin
l	PGSAA_E_GETDBIN	Error in PeVA tool obtaining dbin
u	PGSAA_E_GETSUPP	An error was detected while extracting support data
l	PGSAA_E_POSITION_CALC_FAILURE	The position in the parmBuffer of the requested values was miscalculated
u	PGSAA_E_THREED_READ_ERROR	Function failure to read parameter values from buffer
l	PGSAA_E_EXTRACTORESULTSERROR	Failure to transfer selected values from parmBuffer to results
ul	PGSAA_E_OUTOFRANGE	Input values out of data set range

EXAMPLE:

C:

```
PGSt_SMF_status retStatus;

char parms[PGSs_AA_MAXNOCACHES][100] = {
                          "nmcRucSigPres","nmcRucSigPot"};
      PGSt_integer xStart = 30;
      PGSt_integer yStart = 20;
      PGSt_integer zStart = 2;
      PGSt_integer xDim = 6;
      PGSt_integer yDim = 4;
      PGSt_integer zDim = 2;
      PGSt_integer fileId = 10972;  /* contains interleaved
                                       parms */

float results[2][4][6][2];  /* height,lat,long,parm */
```

```
PGSt_integer      nParms = 2

PGSt_SMF_status = PGS_AA_3DRead (iparms, nParms, xStart,
                                 yStart, zStart, xDim, yDim,
                                 zDim, fileId, 1, 2
                                 results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_3dread
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
integer           xStart
integer           yStart
integer           zStart
integer           xDim
integer           yDim
integer           zDim
integer           fileId
integer           version
integer           operation

integer           result(2,50,20,2)
parms(1)= "nmcRucSigPot"
parms(2)= "nmcRucSigPres"
nParms=2
yStart=102
xStart=205
zStart=2
yDim=20
xDim=50
zDim=2
fileId=10972
operation=2
version = 1

call pgs_aa_3dread( iparms, nparms, xStart, yStart, zStart,
                    xDim, yDim, zDim, fileId, version,
                    operation, results )
```

NOTES: For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or Appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

REQUIREMENTS: PGSTK–0931, PGSTK–1360, PGSTK–1362

6.3.3 Celestial Body Position Tools

The tools included in this section provide the user with information about the locations of celestial bodies (sun, moon, major planets and bright stars). The vector from the Earth or the spacecraft can be computed and the presence of a body in the instrument field of view can be detected.

6.3.3.1 Celestial Body Position Tool Notes

The following notes apply to several of the Celestial Body Position Tools.

TIME RANGE OF CELESTIAL BODY EPHEMERIS

The EOSDIS version of the JPL DE200 ephemeris which is used for the celestial body positions is valid from Dec 14, 1949 through Jan 1, 2021. The user's calling times are internally translated to TDT (dynamical time, similar to the old "ephemeris time") before being used to access the epehemeris itself. This translation depends on leap seconds information. If the leap seconds file is not up to date the error message "PGSTD_NO_LEAP_SECS" is returned but processing continues. Since leap seconds are normally available only six months in advance, results for far future simulations cannot be guaranteed. On the other hand, as time passes, with the leap seconds file properly updated by automatic Toolkit procedures, the positions calculated at any given time, for past times, for the present date, or a few months in advance will be reliable.

TIME OFFSETS:

These functions accept an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'i' of the input data will be processed at the time: (initial time) + (offset[i]).

Examples:

> if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then input[0] will be processed at time 432000.0 and return output[0]
>
> if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then input[0] will be processed at time 432000.0 + offsets[0] and
> return output[0]
>
> if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then each input[i] will be processed at time 432000.0 + offsets[i] and
> the result will be output[i], where i is on the interval [0,N)

ERROR HANDLING:

These functions process data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of these functions will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of these functions will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of these functions will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status level of 'W' does not necessarily mean that some of the data could not be processed. The 'W' level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will have a status level of 'W' to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS above) that processing was not successful at some of the requested times. Therefore in the case of any return status of level 'W', the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

EPHEMERIS AND ATTITUDE DATA QUALITY CONTROL:

Some of the Celestial Body Positioning tools access spacecraft ephemeris and/or attitude data in order to effect their respective transformations. In these cases users may define "masks" for the two data quality flags (ephemeris and attitude) associated with spacecraft ephemeris data. The quality flags are (currently) four byte entities (may be 8 bytes on the cray but only the first four bytes will be considered) that are interpreted bit by bit for meaning (see Section L.3 Quality Flags). Currently the only "fatal" bit (i.e. indicating meaningless data) that will be set prior to access by the Toolkit is bit 16 (where the least significant bit is bit 0). Additionally, the Toolkit will set bit 12 of the quality flag returned for a given user input time if NO data is found for that input time. Note that this usage is different from most of the other bits which indicate the state of some existing data point. By default the Toolkit will set the mask for each of the quality flags to include bit 16 (fatally flawed data) and bit 12 (no data). This means that any data points returned from the tool PGS_EPH_EphemAttit() with an associated quality flag that has either bit 12 or bit 16 set will be rejected by any TOOLKIT function that makes a call to PGS_EPH_EphemAttit() (e.g. these CBP tools) (note that masking is not applied in the tool

PGS_EPH_EphemAttit() itself since users calling this tool directly can examine the quality flags themselves and make their own determination as to which data points to use or reject).

Users may use the Process Control File (PCF) to define their own masks which the Toolkit will then use instead of the defaults mentioned above. The user defined mask should set any bit which the user considers fatal for their purpose (e.g. red limit exceeded). WARNING: if the user defined mask does not have bit 16 set, the Toolkit will pass through data the associated quality flag of which has bit 16 set. The toolkit will not, however, process any data points if the associated quality flag has bit 12 set (i.e. no data exists) whether or not the user mask has bit 12 explicitly set.

Below are the PCF entries which control the value of these masks:

```
# ------------------------------------------------------------------------
# The following parameter is a "mask" for the ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ------------------------------------------------------------------------
# The following parameter is a "mask" for the attitude data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the attitude data quality flag that
# should be considered fatal (i.e. the attitude data associated
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10508|attitude data quality flag mask|65536
```

Note that in the examples above, the value 65536 is the unsigned integer equivalant of a 32 bit binary counter with bits 12 and 16 set. See section 6.2.3 (Process Control Tools) and (Appendix C Process Control Files) for a detailed explanation of the use of the Process Control File.

REFERENCES:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project", Document 445-TP-002-002, May 1995, by P. Noerdlinger.

Compute Earth to Celestial Body ECI Vector

NAME: **PGS_CBP_Earth_CB_Vector( )**

SYNOPSIS:

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_Earth_CB_Vector(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer        cbId,
        PGSt_double         cbVectors[][3])
```

FORTRAN:

```
include'PGS_CBP.f'
include'PGS_TD.f'
include'PGS_SMF.f'
include'PGS_CBP_6.f'
include'PGS_TD_3.f'

integer function
pgs_cbp_earth_cb_vector(numvalues,asciiutc,offsets,cbid,cbvectors)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             cbid
        double precision    cbvectors(3,*)
```

DESCRIPTION: This function computes the Earth–Centered Inertial (ECI J2000) frame vector from the Earth to the selected bodies of Solar System.

INPUTS

Table 6-165. PGS_CBP_Earth_CB_Vector Inputs

NAME	DESCRIPTION	UNITS	MIN	MAX
asciiUTC	UTC time in CCSDS ASCII Time Code A OR B format	time	1961–01–01	see NOTES
offsets	array of offsets of each input UTC time	seconds	see NOTES	see NOTES
cbId	identifier of celestialbody (see list below)	N/A	1	13
numValues	number of required data points 0—only asciiUTC in used any—any time events are used	N/A	0	any

THE DESIGNATION OF THE ASTRONOMICAL BODIES BY CELESTIAL BODY IDENTIFIER (cbId) IS:

cbId =

1 = MERCURY	8 = NEPTUNE
2 = VENUS	9 = PLUTO
3 = EARTH	10 = MOON
4 = MARS	11 = SUN
5 = JUPITER	12 = SOLAR–SYSTEM BARYCENTER
6 = SATURN	13 = EARTH–MOON BARYCENTER
7 = URANUS	

OUTPUTS:

Table 6-166. PGS_CBP_Earth_CB_Vector Outputs

NAME	DESCRIPTION	UNITS	MIN	MAX
cbVectors[][3]	ECI unit vectors from Earth to celestial body first subscript for each time event specified second subscript gives position vector	meter	see NOTES	see NOTES

RETURNS:

Table 6-167. PGS_CBP_Earth_CB_Vector Returns

Return	Description
PGS_S_SUCCESS	Successful completion
PGSCBP_W_EARTH_CB_ID	Earth cbId is specified
PGSCBP_E_INVALID_CB_ID	Invalid celestial body identifier
PGSTD_E_BAD_INITIAL_TIME	Initial input time can not be deciphered
PGSCBP_E_BAD_ARRAY_SIZE	Incorrect array size
PGSCBP_E_UNABLE_TO_OPEN_FILE	Ephemeris file can not be opened
PGSCBP_E_TIME_OUT_OF_RANGE	Initial time is outside the ephemeris bounds
PGSTD_E_NO_LEAP_SECS	No leap second correction available
PGSCBP_W_BAD_CB_VECTOR	One or more errors in CB vectors
PGS_E_TOOLKIT	For unknown errors

EXAMPLES:

C:

```
#define ARRAY_SIZE      3

PGSt_SMF_status   returnStatus;
PGSt_integer      cbId = 10;
PGSt_integer      numValues;
char              asciiUTC[28] = "2002-07-
                  27T11:04:57.987654Z";
PGSt_double       offsets[ARRAY_SIZE] = {3600.0, 7200.0,
                  10800.0};
PGSt_double       cbVectors[ARRAY_SIZE][3];

char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

numValues = ARRAY_SIZE;

returnStatus = PGS_CBP_Earth_CB_Vector(numValues, asciiUTC,
                                       offsets, cbId,
                                       cbVectors)

if (returnStatus != PGS_S_SUCCESS)
{
    PGS_SMF_GetMsg(&returnStatus, err, msg);
    printf ("ERROR: %s\n", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_earth_cb_vector
integer           returnstatus
integer           cbid
integer           numvalues

double precision  offsets(3)
double precision  cbvectors(3,3)

character*27      asciiutc
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = '2002-07-27T11:04:57.987654Z'
cbid = 10
numvalues = 3

returnstatus = pgs_cbp_earth_cb_vector(numvalues, asciiutc,
                                       offsets, cbid,
                                       cbvectors)
```

```
if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err, msg
endif
```

NOTES: See Section 6.3.3.1 Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REQUIREMENTS: PGSTK–0800

Compute Satellite to Celestial Body Vector in Spacecraft Reference Frame

NAME: **PGS_CBP_Sat_CB_Vector( )**

SYNOPSIS:

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_Sat_CB_Vector(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer        cbId,
        PGSt_double         cbVectors[][3])
```

FORTRAN:

```
include 'PGS_CBP.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'
include 'PGS_CBP_6.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'

integer function
pgs_cbp_sat_cbvectors(spacecrafttag,numvalues,asciiutc,offsets,cbid,
                      cbvectors)

        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             cbid
        double precision    cbvectors(3,*)
```

DESCRIPTION: This function computes the vector in the spacecraft reference frame from the spacecraft to the sun, moon, or planets at a given time or range of times.

INPUTS:

Table 6-168. PGS_CBP_Sat_CB_Vector Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of required data points: 0—only asciiUTC is used any—any time events are used	N/A	0	any
asciiUTC [28]	UTC time in CCSDS ASCII Time code A or B format	time	1961–01–01	see NOTES
offsets[]	array of time offsets from asciiUTC in seconds	seconds	see NOTES	see NOTES
cbId	identifier of celestial bodies (see list below)	N/A	N/A	N/A

THE DESIGNATION OF THE ASTRONOMICAL BODIES BY CELESTIAL BODY IDENTIFIER (cbId) IS:

cbId =

1 = MERCURY
2 = VENUS
3 = EARTH
4 = MARS
5 = JUPITER
6 = SATURN
7 = URANUS
8 = NEPTUNE
9 = PLUTO
10 = MOON
11 = SUN
12 = SOLAR–SYSTEM BARYCENTER
13 = EARTH–MOON BARYCENTER

OUTPUTS:

Table 6-169. PGS_CBP_Sat_CB_Vector Outputs

Name	Description	Units	Min	Max
cbVectors[][3]	vectors in spacecraft reference frame from satellite to the celestial body for each time event	meter	see NOTES	see NOTES

RETURNS:

Table 6-170. PGS_CBP_Sat_CB_Vector Returns (1 of 2)

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_W_BELOW_SURFACE	Output vector from ECItoSC below surface
PGSCBP_W_BAD_CB_VECTOR	One or more bad vectors for requested times
PGSCBP_E_BAD_ARRAY_SIZE	numvalues is less than 0
PGSCBP_E_INVALID_CB_ID	Invalid celestial body identifier
PGSMEM_E_NO_MEMORY	Not enough memory for tmpVectors

Table 6-170. PGS_CBP_Sat_CB_Vector Returns (2 of 2)

Return	Description
PGSCBP_E_UNABLE_TO_OPEN_FILE	Unable to open planetary data file
PGSTD_E_BAD_INITIAL_TIME	Initial time is incorrect
PGSCBP_E_TIME_OUT_OF_RANGE	Initial time is outside the ephemeris bounds
PGSTD_E_SC_TAG_UNKNOWN	Invalid spacecraft tag
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times
PGS_E_TOOLKIT	Toolkit error

EXAMPLES:

C:

```
#define ARRAY_SIZE      3

PGSt_SMF_status         returnStatus;

PGSt_integer            numValues = ARRAY_SIZE;

PGSt_double             cbVectors[ARRAY_SIZE][3];
PGSt_double             offsets[ARRAY_SIZE] = {3600.0,
                                    7200.0, 10800.0};

char                    asciiUTC[28] = "2002-07-
                              27T11:04:57.987654Z";
char                    err[PGS_SMF_MAX_MNEMONIC_SIZE];
char                    msg[PGS_SMF_MAX_MSG_SIZE];

returnStatus = PGS_CBP_Sat_CB_Vector(PGSd_EOS_AM, numValues,
                                        asciiUTC, offsets,
                                  PGSd_MOON, cbVectors);

if (returnStatus != PGS_S_SUCCESS)
{
    PGS_SMF_GetMsg(&returnStatus, err, msg);
    printf ("ERROR: %s\n", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_sat_cb_vector
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
integer           cbid
double precision  cbvectors(3,3)
```

```
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = "2002-07-27T11:04:57.987654Z"
cbid = 10
numvalues = 3

returnstatus = pgs_cbp_sat_cb_vector(pgsd_eos_am, numvalues,
>                                    asciiutc, offsets,
>                                    pgsd_moon, cbvector)

if (returnstatus .ne. pgs_s_success) then
     pgs_smf_getmsg(returnstatus, err, msg)
     write(*,*) err, msg
endif
```

NOTES: See Section 6.3.3.1. Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–0680, PGSTK–0810

Get Solar Time and Coordinates

NAME: **PGS_CBP_SolarTimeCoords( )**

SYNOPSIS:

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_SolarTimeCoords(
        char            asciiUTC[28],
        PGSt_double     longitude,
        PGSt_double     *meanSolTimG,
        PGSt_double     *meanSolTimL,
        PGSt_double     *apparSolTimL,
        PGSt_double     *solRA,
        PGSt_double     *solDec)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_cbp_solartimecoords(asciiutc, longitude,
                                         meansoltimg, meansoltiml,
                                         apparsoltiml, solra, soldec)
        character*27    asciiutc
        double precision longitude
        double precision meansoltimg
        double precision meansoltiml
        double precision apparsoltiml
        double precision solra
        double precision soldec
```

DESCRIPTION This tool performs a low accuracy rapid calculation of solar time and coordinates. The accuracy of the equations here is expected to be about 0.5 minutes of time and 0.04 degrees for the coordinates of the sun.

INPUTS:

Table 6-171. PGS_CBP_SolarTimeCoords Inputs

Name	Description	Units	Min	Max
asciiUTC	Coordinated Universal Time in CCSDS ASCII Time Code A or B format	N/A	See NOTES	See NOTES
longitude	longitude of observer (positive is East) Not required for solar coordinates; should be set to 0 in that case	radians	-pi	pi

OUTPUTS:

Table 6-172. PGS_CBP_SolarTimeCoords Outputs

Name	Description	Units	Min	Max
meanSolTimG	Greenwich Mean Solar Time as seconds from midnight	seconds	0	86400
meanSolTimL	Local Mean Solar Time as seconds from midnight	seconds	0	86400
apparSolTimL	Local Apparent Solar Time as seconds from midnight	seconds	0	86400
solRA	Right Ascension of the Mean sun	radians	0	2*pi
solDec	Declination of the Mean sun	radians	-pi	pi

RETURNS:

Table 6-173. PGS_CBP_SolarTimeCoords Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSTD_M_LEAP_SEC_IGNORED	Input leap second has been ignored
PGSTD_E_TIME_FORMAT_ERROR	Error in format of input ASCII UTC time
PGSTD_E_TIME_VALUE_ERROR	Error in value of input ASCII UTC time
PGS_E_TOOLKIT	Something unexpected happened, execution aborted

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       longitude;
PGSt_double       meanSolTimG;
PGSt_double       meanSolTimL;
```

```
PGSt_double       solRA;
PGSt_double       solDec;

strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CBP_SolarTimeCoords(asciiUTC,longitude,
                                       &meanSolTimG,
                                       &meanSolTimL,
                                       &apparSolTimL,
                                       &solRA,&solDec)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
printf("\n longitude: %lf",longitude);

printf("Greenwich Mean Solar Time:%lf  Local Mean Solar
        Time:%lf", meanSolTimG,meanSolTimL);
printf("\n Local Apparent Solar Time:%lf  Solar Right
        Asc/Dec:%lf/%lf", apparSolTimL,solRA,solDec);
```

FORTRAN:

```
implicit none

integer           pgs_cbp_solartimecoords
character*27      asciiutc
double precision  longitude
double precision  meansoltimg
double precision  meansoltiml
double precision  apparsoltiml
double precision  solra
double precision  soldec
integer           returnstatus

asciiutc = '1991-01-01T11:29:30'
longitude = 1.0

returnstatus = pgs_cbp_solartimecoords(asciiutc,longitude,
                                       meansoltimg,
                                       meansoltiml,
                                       apparsoltiml,solra,
                                       soldec)

if(returnstatus .ne. pgs_s_success) go to 90
write(6,*) asciiutc,longitude
write(6,*)meansoltimg,meansoltiml,apparsoltiml,solra,soldec
```

```
90 write(6,99)returnstatus

99 format('ERROR:',I50)
```

NOTES: The equations used in this function are referenced on page C24 of the 1994 Astronomical Almanac. They are low precision formulas that give the apparent coordinates of the sun to a precision of 0.01 degrees and the equation of time to a precision of 0.5 minutes between the years 1950 and 2050. Less accuracy is expected for dates before 1950 and after 2050.

More accurate solar time determination requires improved solar coordinates and the value of UT1–UTC. These items are accessible through other SDP tools.

In particular, the Solar ephemeris yields accurate solar coordinates and the function PGS_TD_gmst() gives Greenwich Mean Sidereal Time. These can be combined to obtain more accurate Mean Solar Time. The difference UT1–UTC is determined within the coordinate system conversion (CSC) group of functions, in the transformations between Earth Centered Rotating (ECR) and Earth Centered Inertial (ECI).

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

REQUIREMENTS: PGSTK–0760

Celestial Body in Field–of–View Indicator

NAME: **PGS_CBP_body_inFOV( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_CBP.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CBP_body_inFOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[][3],
        PGSt_double         *perimFOV_vectors,
        PGSt_tag            cbID,
        PGSt_boolean        inFOVflag[],
        PGSt_double         cb_vector[][3],
        PGSt_double         cb_SCvector[][3])
```

FORTRAN:

```
include 'PGS_TD_3.f'
include 'PGS_CSC_4.f'
include 'PGS_CBP_4.f'
include 'PGS_EPH_4.f'
include 'PGS_MEM_4.f'
include 'PGS_SMF.f'

integer function pgs_cbp_body_infov(numvalues,asciiutc,offsets,
                                    spacecrafttag,numfovperimvec,infovvector,
                                    perimfov_vectors,cbid,infovflag,cb_vector,
                                    cb_scvector)

integer             numvalues
character*27        asciiutc
double precision    offsets(*)
integer             spacecrafttag
integer             numfovperimvec
double precision    infovvector(*)
double precision    perimfov_vectors(3,numfovperimvec,*)
```

integer	cbid
integer	infovflag(*)
double precision	cb_vector(3,*)
double precision	cb_scvector(3,*)

DESCRIPTION: Given a celestial body (CB) identifier (as in the CBP tools) and a field of view (FOV) description, tool returns a flag or flags indicating if the CB is in the FOV, as well as the coordinates of the CB in SC coordinates. Alternatively, the user can specify CB identifier 999 or PGSd_STAR and supply the ECI vector to the body.

INPUTS:

Table 6-174. PGS_CBP_body_inFOV Inputs

Name	Description	Units	Min	Max
numValues	number of time gridpoints	N/A	1	any
asciiUTC	UTC start time	N/A	1979–06–30T00:00:01	2008–01–01T12:00:00
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numFOVperimVec	number of vectors defining FOV perimeter	N/A	3	any
inFOVvector	vector in FOV, in SC coordinates	N/A	N/A	N/A
perimFOV_vectors	vectors in SC coords defining FOV's; MUST be sequential around FOV; middle dimension must be exactly the same value as numFOVperimVec because of the way the array dimensioning works in the function.	N/A	N/A	N/A
cbId	celestial body ID (Earth not included—see PGS_CSC_Earthpt_FOV)	N/A	1	13
cb_vector	ECI vectors of CB (this is an input only when cbId = 999, meaning user input of ECI vector for CB—see notes)	Arbitrary	see PGS_CBP_Earth_CB_Vector	

OUTPUTS:

Table 6-175. PGS_CBP_body_inFOV Outputs

Name	Description	Units	Min	Max
inFOVflag	PGS_TRUE if CB is in FOV—see notes	N/A	N/A	N/A
cb_SCvector	vector of CB in SC coords notes	meters	see PGS_CBP_body_inFOV() notes	

RETURNS:

Table 6-176. PGS_CBP_body_inFOV Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCBP_E_TIME_OUT_OF_RANGE	Initial time is outside the ephemeris bounds
PGSTD_E_BAD_INITIAL_TIME	Initial time is incorrect
PGSCBP_W_BAD_CB_VECTOR	One or more bad vectors for requested times
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSCSC_W_DATA_FILE_MISSING	The data file earthfigure.dat is missing
PGSCBP_E_UNABLE_TO_OPEN_FILE	Unable to open file
PGSCBP_E_INVALID_CB_ID	Invalid celestial body identifier
PGSCBP_W_EARTH_CB_ID	The tool PGS_CSC_Earthpt_FOV() must be used to check for Earth points in the FOV
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSCSC_E_BAD_ARRAY_SIZE	Incorrect array size
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times
PGSEPH_E_NO_DATA_REQUESTED	Both orb and att flags are set to false
PGSCSC_E_INVALID_FOV_DATA	FOV perimeter vectors are invalid
PGSCSC_E_FOV_TOO_LARGE	FOV specification outside algorithmic limits
PGS_E_TOOLKIT	Something unexpected happened

EXAMPLES:

C:

```
#define          ARRAY_SIZE   3
#define          PERIMVEC_SIZE 4
PGSt_SMF_status  returnStatus;
char             asciiUTC[28];
PGSt_double      offsets[ARRAY_SIZE] =
                       {3600.0,7200.0,10800.0};
PGSt_integer     numValues;
PGSt_integer     numFOVperimVec;
```

```
PGSt_double       inFOVvector[ARRAY_SIZE][3] =
                        { {0.0,0.0,100.0},
                          {0.0,0.0,200.0},
                          {0.0,0.0,300.0}
                          };
PGSt_double
      perimFOV_vectors[ARRAY_SIZE][PERIMVEC_SIZE][3]=
                      { {100.0,100.0,100.0},
                        {-100.0,100.0,100.0},
                        {-100.0,-100.0,100.0},
                        {100.0,-100.0,100.0},
                        {200.0,200.0,200.0},
                        {-200.0,200.0,200.0},
                        {-200.0,-200.0,200.0},
                        {200.0,-200.0,200.0},
                        {300.0,200.0,200.0},
                        {-200.0,300.0,200.0},
                        {-200.0,-300.0,300.0},
                        {300.0,-200.0,200.0},
                      };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       cb_SCvector[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");

returnStatus = PGS_CBP_body_inFOV(numValues,asciiUTC,
                                  offsets,PGSd_TRMM,
                                  numFOVperimVec,
                                  inFOVvector,
                                  perimFOV_vectors,
                                  PGSD_MOON,
                                  inFOVflag,NULL,
                                  cb_SCvector);

if(returnStatus != PGS_S_SUCCESS)|
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_body_infov
integer           returnstatus
integer           spacecrafttag
```

```
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
integer           spacecrafttag
integer           numfovperimvec
double precision  infovvector(3,3)
double precision  perimfov_vectors(3,4,3)
integer           cbid
integer           infovflag(3)
double precision  cb_vector(3,3)
double precision  cb_scvector(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

infovvector(1,1) = 0.0
infovvector(1,2) = 0.0
infovvector(1,3) = 0.0

infovvector(2,1) = 0.0
infovvector(2,2) = 0.0
infovvector(2,3) = 0.0

infovvector(3,1) = 0.0
infovvector(3,2) = 0.0
infovvector(3,3) = 0.0

perimfov_vectors(1,1,1) = 100.0
perimfov_vectors(2,1,1) = 100.0
perimfov_vectors(3,1,1) = 100.0

perimfov_vectors(1,2,1) = -100.0
perimfov_vectors(2,2,1) = 100.0
perimfov_vectors(3,2,1) = 100.0

perimfov_vectors(1,3,1) = -100.0
perimfov_vectors(2,3,1) = -100.0
perimfov_vectors(3,3,1) = 100.0

perimfov_vectors(1,4,1) = 100.0
perimfov_vectors(2,4,1) = -100.0
perimfov_vectors(3,4,1) = 100.0

perimfov_vectors(1,1,2) = 200.0
perimfov_vectors(2,1,2) = 200.0
perimfov_vectors(3,1,2) = 200.0
```

```
perimfov_vectors(1,2,2) = -200.0
perimfov_vectors(2,2,2) = 200.0
perimfov_vectors(3,2,2) = 200.0

perimfov_vectors(1,3,2) = -200.0
perimfov_vectors(2,3,2) = -200.0
perimfov_vectors(3,3,2) = 200.0

perimfov_vectors(1,4,2) = 200.0
perimfov_vectors(2,4,2) = -200.0
perimfov_vectors(3,4,2) = 200.0

perimfov_vectors(1,1,3) = 300.0
perimfov_vectors(2,1,3) = 300.0
perimfov_vectors(3,1,3) = 300.0

perimfov_vectors(1,2,3) = -300.0
perimfov_vectors(2,2,3) = 300.0
perimfov_vectors(3,2,3) = 300.0

perimfov_vectors(1,3,3) = -300.0
perimfov_vectors(2,3,3) = -300.0
perimfov_vectors(3,3,3) = 300.0

perimfov_vectors(1,4,3) = 300.0
perimfov_vectors(2,4,3) = -300.0
perimfov_vectors(3,4,3) = 300.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4

returnstatus = pgs_cbp_body_infov(numvalues,asciiutc,
                                  offsets,PGSd_TRMM,
                                  numfovperimvec,
                                  infovvector,
                                  perimfov_vectors,moon,
                                  infovflag,null,
                                  cb_scvector);

if (returnstatus .ne. pgs_s_success) then
     pgs_smf_getmsg(returnstatus, err, msg)
     write(*,*) err, msg
endif
```

NOTES: The FOV is always specified in SC coordinates; for an instrument fixed to the SC, use the same FOV description always; for scanning instruments, user should provide the description appropriate to the scan instant.

numFOVperim must be at least 3. The tool determines if any part of the CB requested lies within the perimeter defined by the vectors perimFOV_vectors[][][3]. The first index in C (last in FORTRAN) is the

time offset index, and the second MUST be sequential around the FOV perimeter. The vector inFOVvector[][3] MUST lie within the FOV. It need not be central, but there will be loss of efficiency if not. The last index in C (first in FORTRAN) on these vectors is for X,Y and Z components in SC coordinates. It is necessary for the user to supply a vector within the FOV for the reason that on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside FOV, which outside.

The vectors "perimFOV_vectors[][][3]" defining the FOV perimeter can be in clock- or counter-clockwise sequence .

The tool may be used on the Sun, Moon, and planets other than the Earth, in which case the cbID must be selected from the standard set (see the tool PGS_CBP_Earth_CB_Vector()). The tool may also be used on another object (such as a star), in which case cbID should be set = 999 and the ECI J2000 coordinates of the star must be supplied in cb_vector[]. The Sun, Moon and planets have finite radii, as specified in the Table below; CB's with cdID = 999 (PGSd_STAR) are assumed to be of negligible radius.

Note on Finite Size of CB: Since a primary use of this tool will be to determine if the Sun, Moon, or a planet intrudes into the FOV, it is important to allow for the finite size of the object. For this purpose, the Moon and Planets are replaced with spheres of the following radii, which are projected on the celestial sphere:

Table 6-177. Physical Radii for CB in FOV Tool

CB	Radius (km)	Explanation
Sun	7 e 5	
Moon	1739	allows for topography
Mercury	2440	
Venus	6055	
Earth	n/a	use tool PGS_CSC_Earthpt_FOV()
Mars	3397	ignore satellites
Jupiter	1890 e 3	include Galilean satellites
Saturn	1225 e 3	include rings, satellites to Titan
Uranus	25600	planet only
Neptune	24800	planet only
Pluto	19600	planet and Charon
999 (STAR)	0.0	a star, or user–defined point

In general, we have included satellites down to the 10th magnitude.

In the case that the celestial body position is invalid for a particular time, then the corresponding cb_SCvector will be set to PGSd_GEO_ERROR_VALUE.

If the CB disk overlaps the FOV only behind the Earth's equatorial bulge and the overlap is barely hidden by it, and the FOV has a sharp corner protruding past the Earth limb it is possible in rare cases that a false positive answer will issue.

See Section 6.3.3.1 Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–0780

6.3.4 Coordinate System Conversion Tools

6.3.4.1 Introduction

The ECI system is J2000. Thus in Fig. 6-2 the Z axis is along the Earth's rotation axis at the epoch of J2000. The ECR system is Earth fixed, i.e. rotating with the Earth. Since its definition includes the effect of polar motion, then in Fig. 6-1 the Z axis is actually along geographic North, which differs very slightly and variably (~ 3 to 15 meters) from the rotational North axis.

6.3.4.2 Unit Vectors for Input

In some functions, a unit vector or a set of unit vectors is required on input. In these cases, the vectors are generally renormalized internally again, anyway, to prevent obscure errors. Thus, generally, any vector defining the correct direction can be supplied; it need not be normalized. An exception is in the transformations between ECI and Spacecraft Coordinates, PGS_CSC_ECItoSC( ) and PGS_CSC_SCtoECI( ). In these functions, the behavior is different for unit vector input and for input vectors in meters. For these two functions, any input vector whose length is between 0.99999 and 1.000001 is assumed to be a unit vector, while any other vector is assumed to be measured in meters.

6.3.4.3 Other Specialized Vectors and Terminology

The vector along a line of sight from the spacecraft may be referred to as a "pixel vector" or "look vector". It could be the boresight of an instrument or it could locate a point in a finite field of view. The "look point" is the intersection of such a vector with the Earth ellipsoid. Vectors designated by the name of a celestial body, such as the "Sun vector" are assumed to point from Earth center or spacecraft center to the celestial body, depending on context. They are in meters unless described as unit vectors. "Latitude" always means geodetic latitude, and the zenith vector at Earth surface is always taken as the normal to the Earth ellipsoid. The "slant range" is from the instrument boresight or spacecraft center to the look point (depending on the accuracy flag). The "field of view" is always defined in spacecraft coordinates. The "subsatellite point" is at the foot of a normal dropped from the spacecraft to the Earth ellipsoid, and its velocity what would be measured by terrestrial instruments on the ellipsoid.

6.3.4.4 Altitudes; Altitude Warnings

In almost all the tools, such as transformations between ECR and Geodetic coordinates, in PGS_CSC_GetFOV_Pixel( ), and PGS_CSC_SubSatPoint( ) the altitude is defined in meters off the Earth ellipsoid, as specified by the user through an Earth ellipsoid tag. If an invalid tag is used, then the WGS84 ellipsoid is used. The altitude for PGS_CSC_ZenithAzimuth( ) is the exception; it must be defined in meters off the *geoid*, because it is used to calculate air density to correct the zenith angle for refraction when that correction is requested. The altitude is ignored otherwise in PGS_CSC_ZenithAzimuth( ). All the functions that input or output altitude, or calculate it internally check for reasonableness. When large negative altitudes are input or generated internally, warning messages issue to the log file, and a warning return will be given

unless there is a more serious problem. The exact depth used to trigger a warning varies according to context. For example, in PGS_CSC_ECItoSC( ), the depth can be as great as 0.02 Earth radii, on the supposition that in an extreme case the user might wish the coordinates of some point that deep in the Earth, while in PGS_CSC_Earthpt_FOV( ) and PGS_CSC_Earthpt_FixedFOV(), the warning is issued if the depth exceeds 50 km. Here, the function is not just a coordinate transformation, but it informs the user if the point can be seen. The 50 km tolerance allows that even if the Earth ellipsoid model is set by the user so large as to include most of the atmosphere, and the Earth point is on the ocean floor, no warning will be returned. For greater depths the point is deemed not to be visible and the answer always PGS_FALSE. The maximum altitude of 100 km is set to include noctilucent clouds. Higher altitudes will be processed, with the answer PGS_TRUE or PGS_FALSE according to the geometry, but a warning is issued. This is the only case in which a large positive altitude results in a warning, because of the context that one is talking about an "Earth point."

6.3.4.5 Lines of sight; visibility of points

The various functions do not check for obstruction of the line of sight by part of the spacecraft or clouds, nor for the occultation of one celestial body by another. The tool PGS_CSC_Earthpt_FOV( ) checks for occultation of the specified point by the solid Earth, i.e., on the far side, and PGS_CBP_body_inFOV( ) checks for occultation by the Earth; in those cases a PGS_FALSE answer is reported.

6.3.4.6 Ranges for variables

The minimum and maximum values specified in the tables are often guidelines only and may not be rigidly enforced. For example, a likely range is indicated for any one component of the spacecraft velocity, but, in principle a velocity component could be anything up to escape velocity. When angles such as latitude or longitude are input, they generally are not be checked against the specified ranges. If they are out of range, the results may be unpredictable. The angles are always in radians. An angle inadvertently supplied in degrees will usually lead to a wrong answer rather than an error return. Various mathematical libraries that vendors supply with compilers may also give degraded performance when given angles that are badly out of range.

6.3.4.7 Updating the UT1 and polar motion file

The file $PGSDAT/CSC/utcpole.dat contains information about UT1 and polar motion used by many tools. Since this information changes with time, the file must be periodically updated. The SDP Toolkit contains utilities to perform this update function. If a new leap second is issued, the data in this file will change for dates after that second. Since the IERS can announce a leap second on as little as 90 days notice, the file will contain data for only 83 days after its last update; this allows time for the posting of a new data set by the U.S. N. O. as described below, and for the running of the Toolkit update. Tools that depend on these data, such as transformations between ECR and ECI, and tools that deliver UT1 or sidereal time, will fail and issue an error return if they are provided input times past the end of the file. The Log Status file will indicate the failure with a message including " PGSTD_E_NO_UT1_VALUE".

The shell script update_utcpole.sh, which is found in $PGSBIN, will update the utcpole.dat file to the current date. To maintain a current utcpole.dat, this script should be run every week, but twice a week is recommended for optimum accuracy (<~ 2m). The U.S. Naval Observatory file, on which the update depends, is normally replaced by a current one by noon, Eastern Standard Time, each Tuesday and Thursday. The accuracy is discussed in Section 6.2.7.5.2. Update_utcpole.sh calls PGS_CSC_UT1_update, a C program that performs most of the actual update work. A Clear Case capable version update_utcpole_CC.sh is provided, as well, with this version of the Toolkit. It must be used from within a Clear Case view belonging to the process owner.

The update is done by collecting the latest information via ftp from United States Naval Observatory in Washington, DC. Their file "finals.data" in the Series 7 directory within server "maia.usno.navy.mil" contains information on UT1-UTC and the x and y pole displacements. The utcpole.dat header contains the date of updating and the file date as listed within ftp for the last "finals.data" used to update it. The function PGS_CSC_UT1_update reformats the new finals.data information and adds it to the utcpole.dat file, overwriting any old information that is superseded. At the DAACs, the process is done automatically by the scheduler. At Science Computing Facilities, for Toolkits through version 5.2.1, drop 4, users will need to have a ".netrc" file in their home directories, as explained in the comments within the scripts. Later releases will not need such a file.

6.3.4.8 Coordinate System Conversion Tool Notes

The following notes apply to several of the Coordinate System Conversion Tools.

TIME OFFSETS:

These functions accept an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'i' of the input data will be processed at the time: (initial time) + (offset[i]). It is recommended that users take advantage of the efficiency that can be gained by processing many time values in one run, using offsets. Many of the tools have been designed to run more efficiently when operating in this mode, and in some cases an internal limit ~30 to 50 has been set on error messaging to the log file in this mode, to prevent excessive growth of the log file.

Examples:

> if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
> then input[0] will be processed at time 432000.0 and return output[0]

if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
then input[0] will be processed at time 432000.0 + offsets[0] and
return output[0]

if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
then each input[i] will be processed at time 432000.0 + offsets[i] and
the result will be output[i], where i is on the interval [0,N)

ERROR HANDLING:

These functions process data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of these functions will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of these functions will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of this function will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status level of 'W' does not necessarily mean that some of the data could not be processed. The 'W' level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will have a status level of 'W' to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS section of the tool of interest) that processing was not successful at some of the requested times. Therefore in the case of any return status of level 'W', the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

EPHEMERIS AND ATTITUDE DATA QUALITY CONTROL:

Many of the Coordinate System Conversion tools access spacecraft ephemeris and/or attitude data in order to effect their respective transformations. In these cases users may define "masks" for the two data quality flags (ephemeris and attitude) associated with spacecraft ephemeris data. The quality flags are (currently) four byte entities (may be 8 bytes on the cray but only the first four bytes will be considered) that are interpreted bit by bit for meaning (see Section L.3 Quality Flags). Currently the only "fatal" bit (i.e. indicating meaningless data) that will be set prior to access by the Toolkit is bit 16 (where the least significant bit is bit 0). Additionally, the Toolkit

will set bit 12 of the quality flag returned for a given user input time if NO data is found for that input time. Note that this usage is different from most of the other bits which indicate the state of some existing data point. By default the Toolkit will set the mask for each of the quality flags to include bit 16 (fatally flawed data) and bit 12 (no data). This means that any data points returned from the tool PGS_EPH_EphemAttit() with an associated quality flag that has either bit 12 or bit 16 set will be rejected by any TOOLKIT function that makes a call to PGS_EPH_EphemAttit() (e.g. these CSC tools) (note that masking is not applied in the tool PGS_EPH_EphemAttit() itself since users calling this tool directly can examine the quality flags themselves and make their own determination as to which data points to use or reject).

Users may use the Process Control File (PCF) to define their own masks which the Toolkit will then use instead of the defaults mentioned above. The user defined mask should set any bit which the user considers fatal for their purpose (e.g. red limit exceeded). WARNING: if the user defined mask does not have bit 16 set, the Toolkit will pass through data the associated quality flag of which has bit 16 set. The toolkit will not, however, process any data points if the associated quality flag has bit 12 set (i.e. no data exists) whether or not the user mask has bit 12 explicitly set.

Below are the PCF entries which control the value of these masks:

```
# ------------------------------------------------------------------------
# The following parameter is a "mask" for the ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ------------------------------------------------------------------------
# The following parameter is a "mask" for the attitude data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the attitude data quality flag that
# should be considered fatal (i.e. the attitude data associated
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10508|attitude data quality flag mask|65536
```

Note that in the examples above, the value 65536 is the unsigned integer equivalant of a 32 bit binary counter with bits 12 and 16 set. See section 6.2.3 (Process Control Tools) and (Appendix C Process Control Files) for a detailed explanation of the use of the Process Control File.

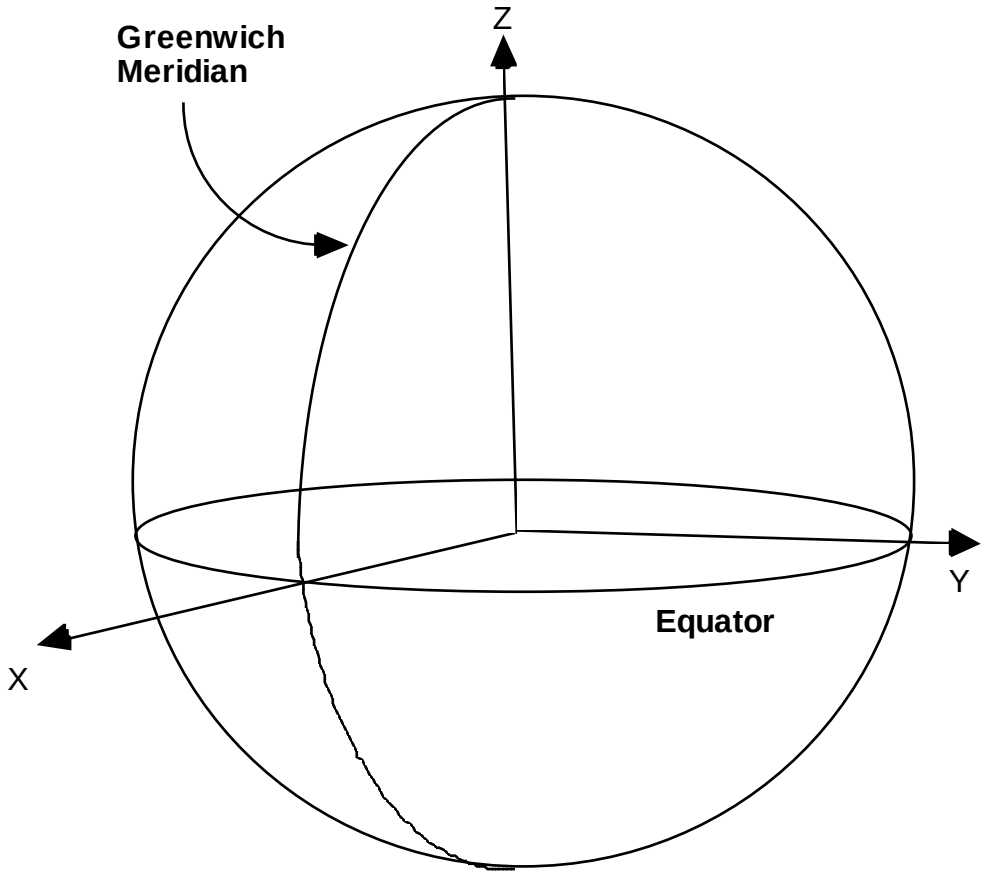
6.3.4.9 Coordinate System Conversion Transformation Tools

These tools convert between various coordinate systems. This will allow calculations to be computed in the most appropriate coordinate system and allow the conversion of results to a common reference frame. Previously these coordinate transformations were contained in one tool

entitled PGS_CSC_FrameChange. We have provided separate calls for each transformation, as one tool proved unwieldy. Also, the user now need not supply extraneous parameters not needed for the desired conversion.

Figures 6–1 through 6–3 show the definitions of the ECR, ECI, and orbital (Orb) reference frames.

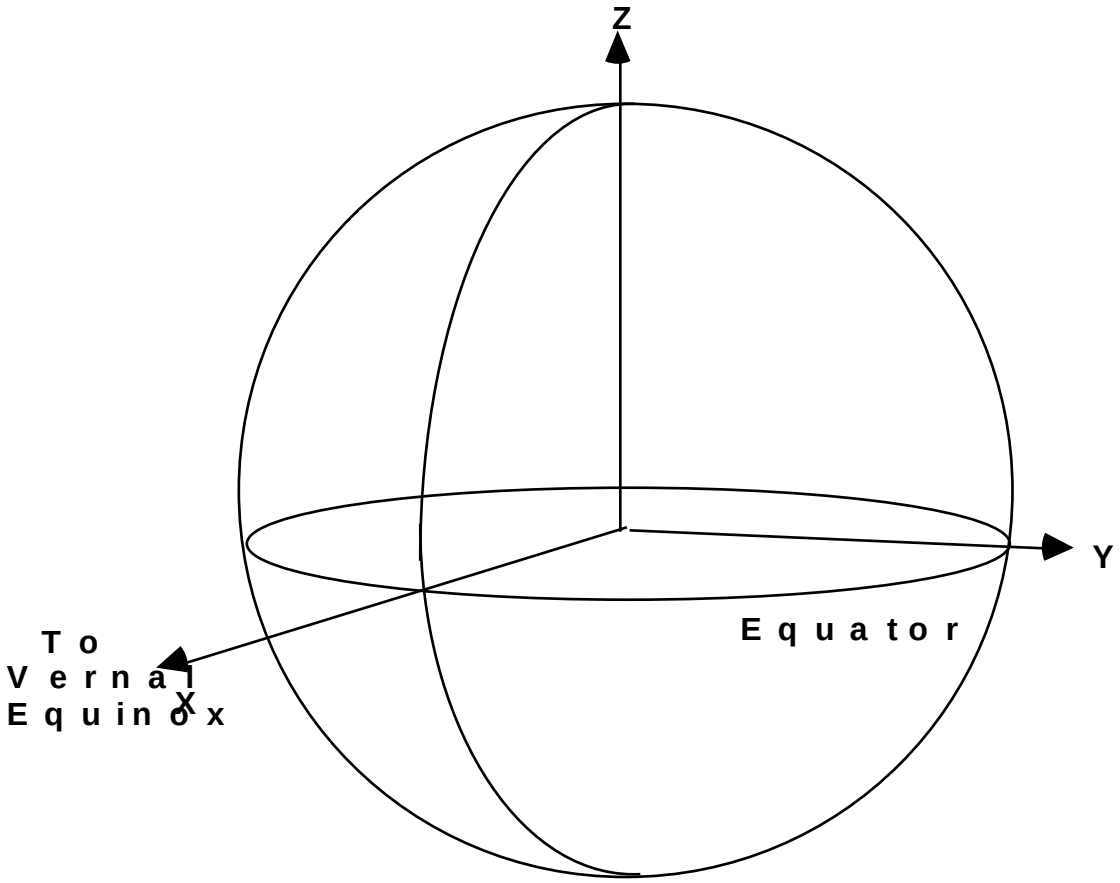
The spacecraft coordinate system coincides with the orbital system when all the Euler angles are zero. Otherwise, it is rotated by the amount indicated by the Euler angles. Thus, a small, positive roll angle indicates that its right side is lowered, and its left side raised. A small positive pitch angle indicates that its nose is raised and thrusters depressed. A small positive zero angle indicates that it is crabbing with its nose to the right of the flight path.



EARTH-CENTERED ROTATING (ECR) COORDINATES

Origin: Center of the Earth
Z-Axis: along Earth's rotational axis, with north positive
X-Y Plane: Earth's equator
X-Axis: directed toward the prime (Greenwich) meridian
Y- Axis: 90 deg from X and Z, completing a right-handed system

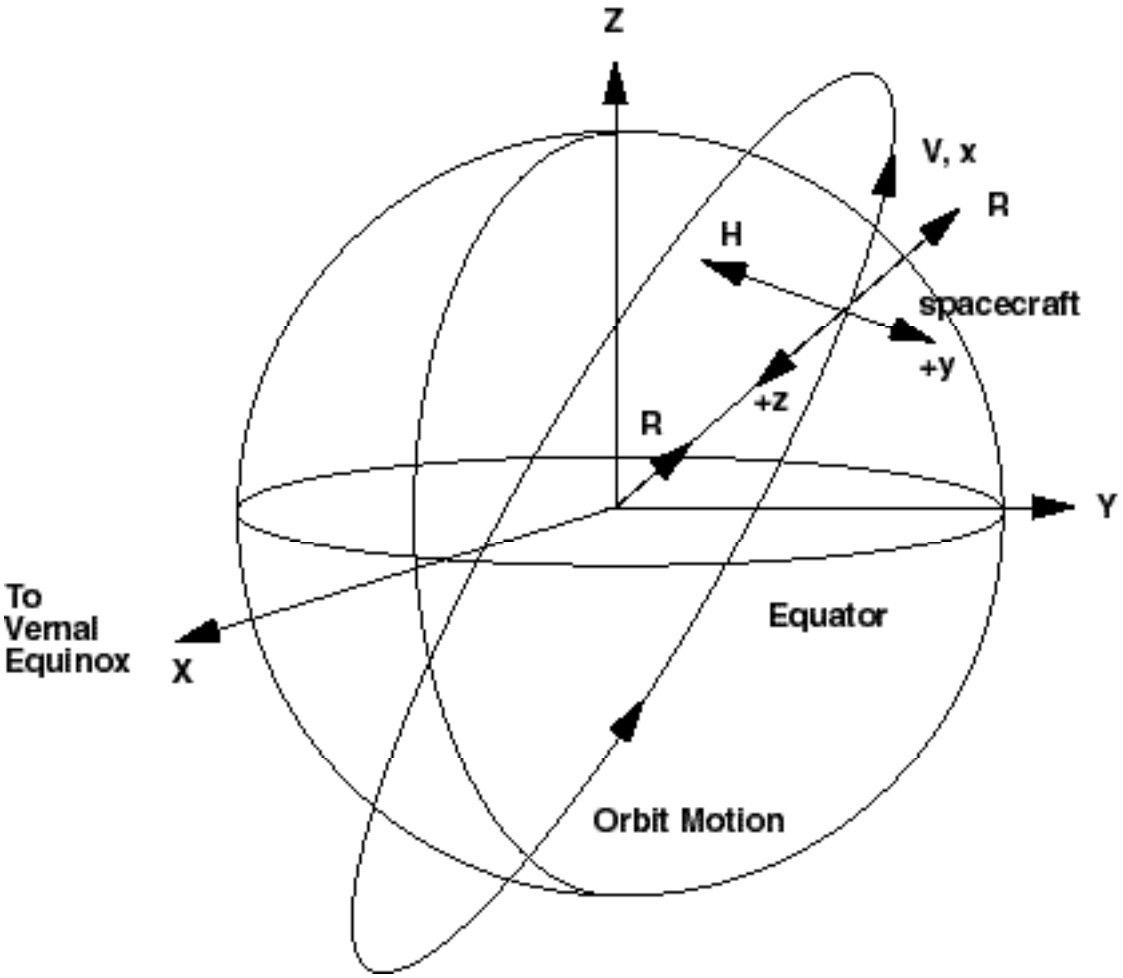
Figure 6-1. Earth-Centered Rotating (ERC) Coordinates



EARTH-CENTERED INERTIAL (

Origin: Center of the Ea
Z-Axis: along Earth's ro
X-Y Plane: Earth's equa
X-Axis: directed toward
Y-Axis: 90 deg from X
system

Figure 6-2. Earth Centered Inertial (ECI) Coordinates



X, Y, Z are inertial coordinates. x, y, z are orbital coordinates, defined as follows:

Origin: Spacecraft Center of Mass
x-z plane: Spacecraft orbital plane

+y (pitch) - Axis: oriented normal to the orbit plane with positive sense opposite to that of the orbit's angular momentum vector H.

+z (yaw) - Axis: positively oriented earthward parallel to the radius vector R from the spacecraft center of mass to the center of the Earth

+x (roll) - Axis: positively oriented in the direction of orbital flight completing an orthogonal triad with y and z.

Figure 6-3. Relationship Between Earth-Centered Inertial (ECI) Coordinates and Orbital Coordinates

Transform from ECI to ECR Coordinates

NAME: **PGS_CSC_ECItoECR( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>

PGSt_SMF_status
PGS_CSC_ECItoECR(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posvelECI[][6],
        PGSt_double         posvelECR[][6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitoecr (numvalues,asciiutc,offsets,posveleci,posvelecr)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posveleci(6,*)
        double precision    posvelecr(6,*)
```

DESCRIPTION: This function rotates an array of 6-vectors from ECI (J2000) coordinates to ECR (of date) coordinates. The rotation is done in 4 parts: precession, nutation, Earth rotation about the nutated axis, and polar motion (correction from the rotational North to geographic North).

INPUTS:

Table 6-178. PGS_CSC_ECItoECR Inputs

Name	Description	Units	Min	Max
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1972-01-01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posvelECI[6]	vector (position and velocity) in J2000 to be transformed to ECR of date			
posvelECI[0]..	x position	meters		
posvelECI[1]	y position	meters		
posvelECI[2]..	z position	meters		
posVelECI[3]	x velocity	meters/ second		
posVelECI[4]	y velocity	meters/ second		
posvelECI[5]	z velocity	meters/ second		

OUTPUTS:

Table 6-179. PGS_CSC_ECItoECR Outputs

Name	Description	Units	Min	Max
posvelECR[0]	vector after being transformed to ECR of date - x position	meters		
posvelECR[1]	vector after being transformed to ECR of date - y position	meters		
posvelECR[2]	vector after being transformed to ECR of date - z position	meters		
posvelECR[3]	vector after being transformed to ECR of date - x velocity	meters/second		
posvelECR[4]	vector after being transformed to ECR of date - y velocity	meters/second		
posvelECR[5]	vector after being transformed to ECR of date - z velocity	meters/second		

RETURNS:

Table 6-180. PGS_CSC_ECItoECR Returns (1 of 2)

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_BAD_TRANSFORM_VALUE	Invalid ECItoECR transformation
PGSCSC_E_BAD_ARRAY_SIZE	Incorrect array size
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available input time
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC

Table 6-180. PGS_CSC_ECItoECR Returns (2 of 2)

Return	Description
PGSCSC_W_PREDICTED_UT1	Status of UT1–UTC correction is predicted
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posvelECI[ARRAY_SIZE][6] =
                                {
                                  {0.5,0.75,0.90,0.3,0.2,0.8},
                                  {0.65,1.2,3.65,0.1,3.2,1,7},
                                  {0.98,2.6,4,78,0.2,1.5,0.9}
                                };
PGSt_double       posvelECR[ARRAY_SIZE][6];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECItoECR(numValues,asciiUTC,offsets,
                                  posvelECI,posvelECR)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
       take appropriate
        action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitoecr
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posvelECI(6,3)
double precision  posvelECR(6,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      asciiutc = '2002-07-27T11:04:57.987654Z'
      numvalues = 3

      DO 10 cnt1 = 1,6
         DO 10 cnt2 = 1,3
               posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 CONTINUE

      returnstatus = pgs_csc_ecitoecr(numValues, asciiutc,
     &                        offsets, posvelECI, posvelECR )

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

NOTES: Users not needing to transform velocity can supply floating point numbers equal to zero for the last three components of each input vector. The Tool cannot transform velocity, however, without correct values for the position. Note that to avoid generating absuredly large velocities for distant objects, no velocity transformation is performed for points more than 500,000,000 m from Earth center.

UTC is: Coordinated Universal Time

J2000 is Julian Date 2451545.0

See Section 6.3.4.8 Coordinate System Conversion Tool Notes.

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES:

The Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

REQUIREMENTS: PGSTK–1050

Transform from ECR to ECI Coordinates

NAME: **PGS_CSC_ECRtoECI( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECRtoECI(
        PGSt_integer    numValues,
        char            asciiUTC[28],
        PGSt_double     offsets[],
        PGSt_double     posvelECR[][6],
        PGSt_double     posvelECI[][6])
```

FORTRAN:

```
include'PGS_CSC_4.f'
include'PGS_TD_3.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function
pgs_csc_ecrtoeci(numvalues,asciiutc,offsets,posvelecr,posveleci)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posvelecr(6,*)
        double precision    posveleci(6,*)
```

DESCRIPTION: This function rotates an array of 6-vectors from ECR (of date) coordinates to ECI (J2000) coordinates. The rotation is done in 4 parts: polar motion (correction from the geographic North to rotational North), rotation about the true rotation, nutation to the mean of date axis axis, and precession to J2000).

INPUTS:

Table 6-181. PGS_CSC_ECRtoECI Inputs

Name	Description	Units	Min	Max
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1972-01-01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posvelECR[6]	vector (position and velocity) in ECR			
posvelECR[0]..	position	meters		
posvelECR[2]				
posvelECR[3]..	velocity	meters/ seconds		
posvelECR[5]				

OUTPUTS:

Table 6-182. PGS_CSC_ECRtoECI Outputs

Name	Description	Units	Min	Max
posvelECI[6]	vector after being transformed to J2000			
posvelECI[0]..	position	meters		
posvelECI[2]		meters		
posvelECI[3]..	velocity	meters/second		
posvelECI[5]		meters/second		

RETURNS:

Table 6-183. PGS_CSC_ECRtoECI Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_BAD_TRANSFORM_VALUE	Invalid ECItoECR transformation
PGSCSC_E_BAD_ARRAY_SIZE	Incorrect array size
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available input time
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSCSC_W_PREDICTED_UT1	Status of UT1–UTC correction is predicted
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posvelECR[ARRAY_SIZE][6] =
                                  {
                                    {0.5,0.75,0.90,0.3,0.2,0.8},
                                    {0.65,1.2,3.65,0.1,3.2,1,7},
                                    {0.98,2.6,4,78,0.2,1.5,0.9}
                                  };
PGSt_double       posvelECI[ARRAY_SIZE][6];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECRtoECI(numValues,asciiUTC,offsets,
                                  posvelECR,posvelECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer       pgs_csc_ecrtoeci
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posveleci(6,3)
double precision  posvelecr(6,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = '2002-07-27T11:04:57.987654Z'
numvalues = 3
```

```
      do 10 cnt1 = 1,6
          do 10 cnt2 = 1,3
              posvelecr(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_ecrtoeci (numValues, asciiutc,
                                       offsets, posvelecr,
                                       posveleci)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

NOTES: Users not needing to transform velocity can supply floating point numbers equal to zero for the last three components of each input vector. The Tool cannot transform velocity, however, without correct values for the position. Note that to avoid generating absuredly large velocities for disant objects, no velocity transformation is performed for points more than 500,000,000 m from Earth center.

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes.

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCES:

The Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

REQUIREMENTS: PGSTK–1050

Convert from ECR to Geodetic Coordinates

NAME: **PGS_CSC_ECRtoGEO( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECRtoGEO(
        PGSt_double     posECR[3],
        char            *earthEllipsTag,
        PGSt_double     *longitude,
        PGSt_double     *latititude,
        PGSt_double     *altitude);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_CSC_4.f'

integer function
pgs_csc_ecrtogeo(posecr,earthellipstag,longitude,latitude,height)
        double precision    posecr(3)
        character*49        earthellipstag
        double precision    longitude
        double precision    latitude
        double precision    altitude
```

DESCRIPTION: This function converts from ECR to geodetic coordinates.

INPUTS:

Table 6-184. PGS_CSC_ECRtoGEO Inputs

Name	Description	Units	Min	Max
posECR[3]	geocentric position	meters	N/A	N/A
EarthEllipsTag	Earth model used	N/A	N/A	N/A

OUTPUTS:

Table 6-185. PGS_CSC_ECRtoGEO Outputs

Name	Description	Units	Min	Max
latitude	geodetic latitude	radians	-pi/2	pi/2
longitude	longitude	radians	-pi	pi
altitude	altitude	meters	-.1* Earth	sky's the limit

RETURNS:

Table 6-186. PGS_CSC_ECRtoGEO Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_TOO_MANY_ITERS	Normal Iteration Count exceeded—could indicate inconsistent units for Spacecraft and Earth data, or corrupted Earth Axis values
PGSCSC_W_INVALID_ALTITUDE	Spacecraft underground—probably indicates bad input data
PGSCSC_W_SPHERE_BODY	Using a spherical Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater then 0.01
PGSCSC_W_DEFAULT_EARTH_MODEL	Uses default Earth model
PGSCSC_E_BAD_EARTH_MODEL	The equatorial or polar radius is negative or zero OR the radii define a prolate Earth
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       longitude
PGSt_double       latitude
PGSt_double       altitude
char              earthEllipsTag[50],
PGSt_double       posECR[3] = {1000.5,64343.56,34343.92}
char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

strcpy(earthEllipsTag,"WGS84");

returnStatus = PGS_CSC_ECRtoGEO(posECR[3],earthEllipsTag,
                                longitude,latitude,
                                altitude);
if(returnStatus != PGS_S_SUCCESS)
  {
      PGS_SMF_GetMsg(&returnStatus,err,msg);
      printf("\nERROR: %s",msg);
  }
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecrtogeo
integer           returnstatus
```

```
double precision  longitude
double precision  latitude
double precision  altitude
character*49      earthellipstag,
double precision  posecr(3)
character*33      err
character*241     msg

data posECR/1000.5,64343.56,34343.92/
earthellipstag = 'WGS84'

returnstatus = pgs_csc_ecrtogeo(posecr,earthellipstag,
                                longitude,latitude,altitude)

if(returnstatus .ne. pgs_s_success) then
    returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
    write(*,*) err, msg
endif
```

NOTES: The Earth axes will be accessed from the earthfigure.dat.file. The input must always be in meters and should never be a unit vector.

REQUIREMENTS: PGSTK–0930, PGSTK–1050

Convert from Geodetic to ECR Coordinates

NAME: **PGS_CSC_GEOtoECR( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GEOtoECR(
        PGSt_double     longitude,
        PGSt_double     latitude,
        PGSt_double     altitude,
        char            *earthEllipsTag,
        PGSt_double     posECR[3]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_CSC_4.f'

integer function
pgs_csc_geotoecr(longitude,latitude,altitude,earthellipstag,posecr)
        double precision    longitude
        double precision    latitude
        double precision    altitude
        character*49        earthellipstag
        double precision    posecr(3)
```

DESCRIPTION: This tool converts a geodetic latitude and longitude to ECR (Earth Centered Rotating) coordinates.

INPUTS:

Table 6-187. PGS_CSC_GEOtoECR Inputs

Name	Description	Units	Min	Max
longitude	longitude	radians	-pi	pi
latitude	latitude	radians	-pi/2	pi/2
altitude	altitude	meters	-.1* radius	N/A
earthellipstag	Earth model used	N/A	N/A	N/A

OUTPUTS:

Table 6-188. PGS_CSC_GEOtoECR Outputs

Name	Description	Units	Min	Max
posECR	ECR rectangular coordinates	meters	-100,000,000 (usually each component will be in range [-10,000,000, +10,000,000 m] but function will work for Geosynchronous cases, e.g.)	100,000,000

RETURNS:

Table 6-189. PGS_CSC_GEOtoECR Returns

Return	Description
PGS_S_SUCCESS	Success case
PGSCSC_W_DEFAULT_EARTH_MODEL	The default Earth model is used because a correct one was not specified
PGSCSC_W_SPHERICAL_BODY	Using a spherical Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater then 0.01
PGSCSC_W_INVALID_ALTITUDE	An invalid altitude was specified
PGSCSC_E_BAD_EARTH_MODEL	The equatorial or polar radius is negative or zero OR the radii define a prolate Earth
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus
PGSt_double       longitude
PGSt_double       latitude
PGSt_double       altitude
char              earthEllipsTag[50],
PGSt_double       posECR[3]
char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

longitude = 0.45;
latitude = 1.34;
altitude = 5000.0;
strcpy(earthEllipsTag,"WGS84");
```

```
returnStatus = PGS_CSC_GEOtoECR(longitude,latitude,altitude,
                                earthEllipsTag,posECR);
if(returnStatus != PGS_S_SUCCESS)
  {
      PGS_SMF_GetMsg(&returnStatus,err,msg);
      printf("\nERROR: %s",msg);
  }
```

FORTRAN:

```
implicit none

integer           pgs_csc_geotoecr
integer           returnstatus
double precision  longitude
double precision  latitude
double precision  altitude
character*49      earthellipstag,
double precision  posecr(3)
character*33      err
character*241     msg

longitude = 0.45
latitude = 1.34
altitude = 5000
earthellipstag = 'WGS84'

returnstatus = pgs_csc_geotoecr(longitude,latitude,altitude,
                                earthellipstag,posecr)

if(returnstatus .ne. pgs_s_success) then
    returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
    write(*,*) err, msg
endif
```

NOTES: NONE

REQUIREMENTS: PGSTK–0930, PGSTK–1050

Transform from ECI Frame to Spacecraft Reference Frame

NAME: **PGS_CSC_ECItoSC( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECItoSC(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posECI[][3],
        PGSt_double         posSC[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitosc(spacecraftTag,numvalues,asciiutc,offsets,poseci,possc)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    poseci(3,*)
        double precision    possc(3,*)
```

DESCRIPTION: Transforms vector in ECI coordinate system to vector in Spacecraft coordinate system. If a unit vector is input, only its direction is transformed. If a vector in meters is input, it is first corrected for the displacement between Earth center and spacecraft location and then rotated into spacecraft coordinates.

INPUTS:

Table 6-190. PGS_CSC_ECItoSC Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1961–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posECI	coordinates or unit vector components in ECI reference frame	meter	N/A	N/A

OUTPUTS:

Table 6-191. PGS_CSC_ECItoSC Outputs

Name	Description	Units	Min	Max
posSC	coordinates or unit vector components in spacecraft reference frame	meters	N/A	N/A

RETURNS:

Table 6-192. PGS_CSC_ECItoSC Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times
PGSEPH_E_NO_DATA_REQUESTED	Both orb and att flags are set to false

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posECI[ARRAY_SIZE][3] =
                                  {
                                    {0.5,0.75,0.90,0.3,0.2,0.8},
                                    {0.65,1.2,3.65,0.1,3.2,1,7},
                                    {0.98,2.6,4,78,0.2,1.5,0.9}
                                  };
PGSt_double       posSC[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECItoSC(PGSd_TRMM,numValues,asciiUTC,
                                offsets,posECI,posSC)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitosc
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  poseci(3,3)
double precision  possc(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cnt1 = 1,3
    do 10 cnt2 = 1,3
        posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_ecitosc(PGSd_TRMM, numValues,
   asciiutc,
                                  offsets, poseci, possc)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

NOTES: Points are checked to make sure they are not subterranean, but no other visibility check is performed (such as line–of–sight).

Next the function checks the input vector to see if it is a unit vector. If so, it is assumed that the user wishes only to transform its direction. If not, it is assumed that the vector locates some point of interest (for example, a TDRSS satellite, or a lookpoint). Thus, for that case a translation to the spacecraft center is performed first and then a rotation. Aberration correction is also performed in both cases, except in the second case, for points within 120 m of spacecraft center. Vectors to such points are not aberrated. This cutoff is imposed on the supposition that anyone wishing to transform a point within 120 m of the spacecraft center could be dealing with an alignment, glint, or other spacecraft-related problem, in which case there is no aberration. For the purposes of this function, a vector is a unit vector if its magnitude is between 0.99999 and 1.00001.

TIME ACRONYMS:

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

Transform Between Spacecraft and ECI Reference Frames

NAME: **PGS_CSC_SCtoECI( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SCtoECI(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posSC[][3],
        PGSt_double         posECI[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_sctoeci(spacecraftTag,numvalues,asciiutc,offsets,possc,poseci)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    possc(3,*)
        double precision    poseci(3,*)
```

DESCRIPTION: Transforms vector in Spacecraft coordinate system to vector in ECI coordinate system. If a unit vector is input, it is simply rotated to ECI coordinates. If a vector in meters in input, it is rotated to ECI axes and then translated from having its origin at the spacecraft center to having its origin at Earth center.

INPUTS:

Table 6-193. PGS_CSC_SCtoECI Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1961–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posSC	coordinates or unit vector components in SC reference frame	meters	N/A	N/A

OUTPUTS:

Table 6-194. PGS_CSC_SCtoECI Outputs

Name	Description	Units	Min	Max
posECI	coordinates or unit vector components in ECI reference frame	meters	N/A	N/A

RETURNS:

Table 6-195. PGS_CSC_SCtoECI Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input
PGSEPH_E_NO_DATA_REQUESTED	Both orb and att flags are set to false

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posSC[ARRAY_SIZE][3] =
                                  {
                                    {0.5,0.75,0.90,0.3,0.2,0.8},
                                    {0.65,1.2,3.65,0.1,3.2,1,7},
                                    {0.98,2.6,4,78,0.2,1.5,0.9}
                                  };
PGSt_double       posECI[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_SCtoECI(PGSd_TRMM,numValues,
                                asciiUTC,offsets, posSC,
                                posECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer                 pgs_csc_sctoeci
integer                 returnstatus
integer                 numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  possc(3,3)
double precision  poseci(3,3)
integer                 cnt1
integer                 cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_sctoeci (PGSd_TRMM,numValues,
                                      asciiutc,offsets,possc,
                                      poseci)

      if (returnstatus .ne. pgs_s_success) then
           pgs_smf_getmsg(returnstatus, err, msg)
           write(*,*) err, msg
      endif
```

NOTES: This function first checks the input vector to see if it is a unit vector. If so, it is assumed that the user wishes only to transform its direction. If not, it is assumed that the vector locates some point of interest (for example, a TDRSS satellite, or a lookpoint). For that case a rotation to ECI axes is performed first, and then a translation to the Earth center. An aberration correction is also made if the input is a unit vector or is in meters and represents a point more than 120 m from spacecraft center. This cutoff is imposed on the supposition that anyone wishing to transform a point within 120 m of the spacecraft center could be dealing with an alignment, glint, or other spacecraft-related problem, in which case there is no aberration. For the purposes of this function, a vector is a unit vector if its magnitude is between 0.99999 and 1.00001

Certain checks are performed in the case of translation to ensure that the transformed point is not below the Earth's surface; other visibility checks (such as line–of–sight) are not performed.

TIME ACRONYMS:

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

Transform from Spacecraft Frame to Orbital Frame

NAME: **PGS_CSC_SCtoORB( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SCtoORB(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posSC][3],
        PGSt_double         posORB[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_sctoorb(spacecraftTag,numvalues,asciiutc,offsets,possc,posorb)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    possc(3,*)
        double precision    posorb(3,*)
```

DESCRIPTION: Transforms vector in Spacecraft reference frame to a vector in Orbital reference frame.

INPUTS:

Table 6-196. PGS_CSC_SCtoORB Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1961–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posSC	coordinates or unit vector components in SC reference frame	meters	N/A	N/A

OUTPUTS:

Table 6-197. PGS_CSC_SCtoORB Outputs

Name	Description	Units	Min	Max
posORB	coordinates or unit vector components in Orbital reference frame	meters	N/A	N/A

RETURNS:

Table 6-198. PGS_CSC_SCtoORB Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posSC[ARRAY_SIZE][3] =
                                  {
                                    {0.5,0.75,0.90,0.3,0.2,0.8},
                                    {0.65,1.2,3.65,0.1,3.2,1,7},
                                    {0.98,2.6,4,78,0.2,1.5,0.9}
                                  };
PGSt_double       posORB[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_SCtoORB(pgsd_trmm,numvalues,asciiutc,
                                offsets,possc,posORB)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_sctoorb
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  possc(3,3)
double precision  posorb(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cbt1 = 1,3
    do 10 cnt2 = 1,3
        possc(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_sctoorb(pgsd_trmm,numvalues,
   asciiutc,
                                  offsets, possc, posorb)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

NOTES: **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

Transform from Orbital Frame to Spacecraft Frame

NAME: **PGS_CSC_ORBtoSC( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ORBtoSC(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posORB][3],
        PGSt_double         posSC[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_orbtosc(spacecrafttag,numvalues,asciiutc,offsets,posorb,possc)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posorb(3,*)
        double precision    possc(3,*)
```

DESCRIPTION: Transforms vector from Orbital reference frame to a vector in Spacecraft reference frame.

INPUTS:

Table 6-199. PGS_CSC_ORBtoSC Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1960–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
posORB	coordinates or unit vector components in Orbital reference frame	meters	N/A	N/A

OUTPUTS:

Table 6-200. PGS_CSC_ORBtoSC Outputs

Name	Description	Units	Min	Max
posSC	coordinates or unit vector components in spacecraft reference frame	meters	N/A	N/A

RETURNS:

Table 6-201. PGS_CSC_ORBtoSC Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posORB[ARRAY_SIZE][3] =
                                  {
                                    {0.5,0.75,0.90,0.3,0.2,0.8},
                                    {0.65,1.2,3.65,0.1,3.2,1,7},
                                    {0.98,2.6,4,78,0.2,1.5,0.9}
                                  };
PGSt_double       posSC[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ORBtoSC(PGSd_TRMM,numValues,asciiUTC,
                                  offsets, posORB, posSC)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_orbtosc
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posorb(3,3)
double precision  possc(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cnt1 = 1,3
    do 10 cnt2 = 1,3
        posorb(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_orbtosc(pgsd_trmm,numvalues,
   asciiutc,
                                  offsets, posorb, possc)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

NOTES: **TIME ACRONYMS:**

UTC is: Coordinted Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

Transform from ECI Frame to Orbital Frame

NAME: **PGS_CSC_ECItoORB( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECItoORB(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         positionECI[][3],
        PGSt_double         positionORB[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EHP_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitoorb(spacecraftTag,numvalues,asciiutc,offsets,positioneci,
                                                        positionorb)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    positioneci(3,*)
        double precision    positionorb(3,*)
```

DESCRIPTION: Transforms vector in ECI coordinate system to vector in Orbital coordinate system. If a unit vector is input only its direction is changed. If a vector in meters is input, it is first translated from the Earth centered system to a spacecraft centered origin, and then rotated to orbital coordinate axes.

INPUTS:

Table 6-202. PGS_CSC_ECItoORB Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1961–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
positionECI	coordinates or unit vector components in ECI reference frame	meters	N/A	N/A

OUTPUTS:

Table 6-203. PGS_CSC_ECItoORB Outputs

Name	Description	Units	Min	Max
positionORB	coordinates or unit vector components in orbital reference frame	meters	N/A	N/A

RETURNS:

Table 6-204. PGS_CSC_ECItoORB Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                              {3600.0,7200.0,10800.0};
PGSt_double       positionORB[ARRAY_SIZE][3] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       positionORB[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus =
PGS_CSC_ECItoORB(PGSd_TRMM,numValues,asciiUTC,
                                  offsets, positionECI,
                                  positionORB)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitoorb
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  positioneci(3,3)
double precision  positionorb(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              positioneci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_ecitoorb(pgsd_trmm,numvalues,
                                      asciiutc,offsets,
                                      positioneci, positionorb)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

NOTES: **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

Transform from Orbital Frame to ECI Frame

NAME: **PGS_CSC_ORBtoECI( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ORBtoECI(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         positionORB[][3],
        PGSt_double         positionECI[][3])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_SMF.f'

integer function pgs_csc_orbtoeci(spacecraftTag,numvalues,asciiutc,
                                  offsets,positionorb,positioneci)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    positionorb(3,*)
        double precision    positioneci(3,*)
```

DESCRIPTION: Transforms vector in Orbital coordinate system to vector in ECI coordinate system. If a unit vector is input it is simply rotated from Orbital to ECI axes. If a vector in meters is input, it is first rotated from Orbital to ECI axes and then translated from the system referenced at spacecraft center to the system referenced at Earth center.

INPUTS:

Table 6-205. PGS_CSC_ORBtoECI Inputs

Name	Description	Units	Min	Max
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1961–01–01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
positionORB	coordinates or unit vector components in Orbital reference frame	meters	N/A	N/A

OUTPUTS:

Table 6-206. PGS_CSC_ORBtoECI Outputs

Name	Description	Units	Min	Max
positionECI	coordinates or unit vector components in ECI reference frame	meters	N/A	N/A

RETURNS:

Table 6-207. PGS_CSC_ORBtoECI Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSSC_W_BELOW_SURFACE	Vector magnitude indicates subsurface location specified
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_TIME_FMT_ERROR	Format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Value error in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                                {3600.0,7200.0,10800.0};
PGSt_double       positionORB[ARRAY_SIZE][3] =
                                {
                                  {0.5,0.75,0.90,0.3,0.2,0.8},
                                  {0.65,1.2,3.65,0.1,3.2,1,7},
                                  {0.98,2.6,4,78,0.2,1.5,0.9}
                                };
PGSt_double       positionECI[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus =
PGS_CSC_ORBtoECI(PGSd_TRMM,numValues,asciiUTC,offsets,
                 positionORB,positionECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_orbtoeci
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  positionorb(3,3)
double precision  positioneci(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              positionorb(cnt1,cnt2) = 100 * cnt1 * cnt2
   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_orbtoeci(pgsd_trmm,numvalues,
                                      asciiutc,offsets,
                                      positionorb,positioneci)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

NOTES: **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1050

6.3.4.10 Coordinate System Conversion—Other Tools

These tools provide other location and orientation information to the user.

Get Sub–Satellite Point Position and Velocity

NAME: **PGS_CSC_SubSatPoint( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SubSatPoint(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        char                earthEllipsTag[50],
        PGSt_boolean        velFlag,
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_double         altitude[],
        PGSt_double         velSub[][3])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_MEM_7.f'

integer function
pgs_csc_subsatpoint(spacecrafttag,numvalues,asciiutc,offsets,

        earthellipstag,velflag,latitude,longitude,
                                altitude,velsub)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        character*49        earthellipstag
        integer             velflag
        double precision    latitude(*)
```

double precision	longitude(*)
double precision	altitude(*)
integer	velsub(3,*)

DESCRIPTION: This tool finds the latitude, longitude, and altitude of the subsatellite points at the input times/offsets and, optionally, returns North and East components of each subsatellite point. The third component returned for each subsatellite point, when velocity is requested, is the rate of change of the spacecraft altitude off the Earth ellipsoid (as would be measured by a Doppler radar altimiter, ignoring terrain).

INPUTS:

Table 6-208. PGS_CSC_SubSatPoint Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A	N/A	N/A
numValues	number of input offset times	N/A	0	any
asciiUTC	timesstart UTC time in CCSDS ASCII Time Code (A or B format)	N/A	1979–06–30	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC + offset is between Min and Max values	
earthEllipsTag	tag selecting Earth ellipsoid model (default is WGS84)	N/A	N/A	N/A
velFlag	flag indicating whether to return the velocity of the subsatellite points	N/A	PGS_FALSE	PGS_TRUE

OUTPUTS:

Table 6-209. PGS_CSC_SubSatPoint Outputs

Name	Description	Units	Min	Max
latitude	array of subsatellite point geodetic latitudes	radians	-pi/2	pi/2
longitude	array of subsatellite point longitudes	radians	-pi	pi
altitude	array of spacecraft altitudes	m	250000	10000000
velSub[0]	North component of the subsatellite point velocity on the ellipsoid	m/s	-7000	7000
velSub[1]	East component of the subsatellite point velocity on the ellipsoid	m/s	-7000	7000
velSub[2]	rate of change of spacecraft altitude relative to nadir on the ellipsoid	m/s	-200	200

RETURNS:

Table 6-210. PGS_CSC_SubSatPoint Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_ERROR_IN_SUBSATPT	An error occurred in computing at least one subsatellite point
PGSCSC_W_PREDICTED_UT1	At least one of the values obtained from the utcpole.dat file is 'predicted'
PGSCSC_W_PROLATE_BODY	Using a prolate Earth model
PGSCSC_W_SPHERE_BODY	Using a spherical Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater than 0.01
PGSCSC_W_DEFAULT_EARTH_MODEL	Default Earth model was used
PGSCSC_W_ZERO_JACOBIAN_DET	Jacobian determinant is close to zero
PGSCSC_E_BAD_ARRAY_SIZE	numValues (and array size) is less than zero
PGSMEM_E_NO_MEMORY	No memory available to allocate vectors
PGSTD_E_SC_TAG_UNKNOWN	Invalid spacecraft tag
PGSEPH_E_BAD_EPHEM_FILE_HEADER	No spacecraft ephemeris files had reasonable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No spacecraft ephemeris files could be found for input
PGSTD_E_TIME_FMT_ERROR	Format error in input asciiUTC
PGSTD_E_TIME_VALUE_ERROR	Error in one of time values in asciiUTC
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for at least one of the input times/offsets—a linear approximation was used to obtain the leapsec value
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGSTD_E_BAD_EARTH_MODEL	The equatorial or polar radius is negative or zero OR the radii define a prolate Earth
PGS_E_TOOLKIT	Something unexpected happened—execution aborted

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_tag          spacecraftTag = PGSd_EOS_AM;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                  {3600.0,7200.0,10800.0};
char              earthEllipsTag[50];
PGSt_boolean      velFlag = PGS_TRUE;
PGSt_double       latitude[ARRAY_SIZE];
```

```
PGSt_double       longitude[ARRAY_SIZE];
PGSt_double       altitude[ARRAY_SIZE];
PGSt_double       velSub[ARRAY_SIZE][3];
PGSt_integer      counter;

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
strcpy(earthEllipsTag,"WGS84");

returnStatus = PGS_CSC_SubSatPoint(spacecraftTag,numValues,
                                   asciiUTC,offsets,
                                   earthEllipseTag,velFlag,
                                   latitude,longitude,
                                   altitude,velSub);

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter <= numValues)
{
    printf("Offset:  %lf   Latitude:  %lf   Longitude:  %lf
            Altitude:  %lf",   offset[counter],
            latitude[counter],  longitude[counter],
            altitude[counter]);
    printf("Velocity of subsatellite point
            (North,East,altitude): %lf, %lf, %lf"  " m/s",
             velSub[counter][0], velSub[counter][1],
             velSub[counter][2]);

             counter++;
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_subsatpoint
integer           array_size
integer           spacecrafttag
integer           numvalues
character*27      asciiutc
double precision  offsets(array_size)
character*49      earthellipstag
integer           velflag
```

```
      double precision  latitude(array_size)
      double precision  longitude(array_size)
      double precision  altitude(array_size)
      double precision  velsub(3,array_size)
      integer           returnstatus
      integer           counter

      data offsets/3600.0,7200.0,10800.0/
      data earthellipstag/'WGS84'/,velflag/PGS_TRUE/
      array_size = 3
      numvalues = array_size
      spacecrafttag = pgsd_eos_am
      asciiutc = '1991-01-01T11:29:30'

      returnstatus = pgs_csc_subsatpoint(spacecrafttag,numvalues,
                                         asciiutc,offsets,
                                         earthellipsetag,velflag,
                                         latitude,longitude,
                                         altitude,velsub)

      if(returnstatus .ne. pgs_s_success) go to 90
      write(6,*) asciiutc
      do 40 counter = 0,numvalues,1
          write(6,*)offsets(counter), latitude(counter),
                    longitude(counter), altitude(counter),
                    velsub(1,counter), velsub(2,counter),
                    velsub(3,counter)

   40 continue

   90 write(6,99)returnstatus

   99 format('ERROR:',I50)
```

NOTES:

If an error occurs during computation for one or more input times but does not necessarily affect all input times, latitude, longitude, altitude, and velocity values of PGSd_GEO_ERROR_VALUE are returned for the input times where the error occurred. An indication that an error occurred in this tool is returned in the returnStatus value, and a description of the error is returned in the corresponding message.

If an invalid earthEllipsTag is input, the program will use the WGS84 Earth model by default.

The option to obtain velocity is controlled by setting the velocity flag velFlag to either PGS_TRUE or PGS_FALSE. If velFlag is PGS_FALSE, all components of velSub will be set to zero. If the velocity is not needed it is recommended to use PGS_FALSE to speed the execution of the code.

The horizontal velocity calculated in function PGS_CSC_SubSatPointVel() is that of a mathematical point on the Earth at (nominal) spacecraft nadir, and not that of any material object. It is orthogonal to nadir, so is suitable as a descriptor of ground track but not for Doppler work.

The third (vertical) component of velocity is useful for Doppler work at nadir, but Doppler velocity along ANY look vector (not just nadir) is provided in the lookpoint algorithm in the function PGS_CSC_GetFOV_Pixel( ).

The condition PGSCSC_W_ZERO_JACOBIAN_DET is not expected to occur. Its appearance would indicate that the geometry is singular: the altitude of the spacecraft is zero or the spacecraft is exactly at the north or south pole, for example.

TIME ACRONYMS:

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REFERENCES FOR TIME:

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK–0930, PGSTK–1060

Get Times of Earth Point in Fixed Field of View

NAME: **PGS_CSC_Earthpt_FixedFOV( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CSC_Earthpt_FixedFOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        char                *earthEllipsTag,
        PGSt_double         latitude,
        PGSt_double         longitude,
        PGSt_double         altitude,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[3],
        PGSt_double         perimFOV_vectors[][3],
        PGSt_boolean        inFOVflag[],
        PGSt_double         sctoEarthptVec[][3])
```

FORTRAN:

```
include'PGS_TD_3.f'
include'PGS_CSC_4.f'
include'PGS_EPH_5.f'
include'PGS_MEM_7.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function pgs_csc_earthpt_fixedfov(numvalues,asciiutc,offsets,
                                          spacecrafttag,earthellipstag,latitude,
                                          longitude,altitude,numfovperimvec,
                                          infovvector,perimfov_vectors,
                                          infovflag,sctoearthptvec)

        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             spacecrafttag
        character*49        earthellipstag
```

```
double precision    latitude
double precision    longitude
double precision    altitude
integer             numfovperimvec
double precision    infovvector(3)
double precision    perimfov_vectors(3,*)
integer             infovflag(*)
double precision    sctoearthptvec(3,*)
```

DESCRIPTION: For each time value, the tool, using the FOV description, returns a flag or flags indicating if the Earth point of given latitude, longitude and altitude is in the FOV, and the vector to that point from the SC in SC coordinates.

INPUTS:

Table 6-211. PGS_CSC_Earthpt_FixedFOV Inputs

Name	Description	Units	Min	Max
numValues	number of time gridpoints	N/A	0	any
asciiUTC	UTC start time	N/A	1972-01-01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
earthEllipsTag	Earth model used	N/A	N/A	N/A
latitude	latitude of Earth point	radians	-pi/2	+pi/2
longitude	longitude of Earth point	radians	-2*pi	+2*pi
altitude	altitude of Earth point	meters	-50000	100000
numFOVperimVec	number of vectors defining FOV perimeter	N/A	3	any
inFOVvector	vector in FOV—preferably near the center in SC coordinates	N/A	N/A	N/A
perimFOV_vectors	vectors in SC coords defining FOV's; MUST be sequential around FOV; the middle dimension must be exactly the same as numFOVperimVec because of the way the array dimensioning works in the function	N/A	N/A	N/A

OUTPUTS:

Table 6-212. PGS_CSC_Earthpt_FixedFOV Outputs

Name	Description	Units	Min	Max
inFOVflag	PGS_TRUE if Earth point is in FOV—see notes	n/a	n/a	n/a
sctoEarthptVec	vector to Earth point in SC coords—returned normalized	meters	-1	1

RETURNS:

Table 6-213. PGS_CSC_Earthpt_FixedFOV Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	Location is below surface
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_BAD_INITIAL_TIME	Initial time is incorrect
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSCSC_W_DEFAULT_EARTH_MODEL	The default Earth model is used because a correct one was not specified
PGSCSC_W_DATA_FILE_MISSING	The data file earthfigure.dat is missing
PGSCSC_W_SPHERICAL_BODY	Using a spherical Earth model
PGSCSC_W_PROLATE_BODY	Using a prolate Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater then 0.01
PGSCSC_E_INVALID_ALTITUDE	An invalid altitude was specified
PGSCSC_E_NEG_OR_ZERO_RAD	The equatorial or polar radius is negative or zero
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSCSC_E_BAD_ARRAY_SIZE	Incorrect array size
PGSCSC_W_PREDICTED_UT1	Status of UT1–UTC correction is predicted
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times
PGSCSC_E_INVALID_FOV_DATA	FOV perimeter vectors are invalid
PGSCSC_E_FOV_TOO_LARGE	FOV specification outside algorithmic limits
PGSCSC_E_INVALID_EARTH_PT	One of the Earth point vectors was zero
PGSCSC_W_ZERO_PIXEL_VECTOR	Instrument pixel vector of zero length
PGSCSC_W_BAD_EPH_FOR_PIXEL	Ephemeris Data missing for some pixels

EXAMPLES:

C:

```
#define   ARRAY_SIZE   3
#define   PERIMVEC_SIZE 4
```

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_integer      numValues;
PGSt_double       latitude;
PGSt_double       longitude;
PGSt_double       altitude;
PGSt_integer      numFOVperimVec;
PGSt_double       inFOVvector[3] =
                        { {0.0,0.0,100.0},
                        };
PGSt_double       perimFOV_vectors[PERIMVEC_SIZE][3]=
                        { {100.0,100.0,100.0},
                          {-100.0,100.0,100.0},
                          {-100.0,-100.0,100.0},
                          {100.0,-100.0,100.0} };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       sctoEarthptVec[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");
altitude = 10000.0;
latitude = 0.32;
longitude = 2.333;
returnStatus =
PGS_CSC_Earthpt_FixedFOV(numValues,asciiUTC,offsets,
                   PGSd_TRMM,"WGS84",latitude,
                   longitude,altitude,numFOVperimVec,
                   inFOVvector,perimFOV_vectors,inFOVflag,
                   sctoEarthptVec)
if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_earthpt_fixedfov
integer           returnstatus
integer           numvalues
character*27      startutc
double precision  offsets(3)
double precision  latitude
double precision  longitude
double precision  altitude
integer           numfovperimvec
double precision  infovvector(3)
double precision  perimfov_vectors(3,4)
integer           infovflag(3)
double precision  sctoearthptvec(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

perimfov_vectors(1,1) = 100.0
perimfov_vectors(2,1) = 100.0
perimfov_vectors(3,1) = 100.0

perimfov_vectors(1,2) = -100.0
perimfov_vectors(2,2) = 100.0
perimfov_vectors(3,2) = 100.0

perimfov_vectors(1,3) = -100.0
perimfov_vectors(2,3) = -100.0
perimfov_vectors(3,3) = 100.0

perimfov_vectors(1,4) = 100.0
perimfov_vectors(2,4) = -100.0
perimfov_vectors(3,4) = 100.0

infovvector(1) = 0.0
infovvector(2) = 0.0
infovvector(3) = 100.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4
altitude = 10000.0
latitude = 0.32
longitude = 2.333
```

```
returnstatus =
pgs_csc_earthpt_fixedfov(numvalues,startutc,offsets,
                    PGSd_TRMM,'WGS84',latitude,
                    longitude,altitude,numfovperimvec,
                    infovvector,perimfov_vectors,
                    infovflag,sctoearthptvec)

if (returnstatus .ne. pgs_s_success) then
    pgs_smf_getmsg(returnstatus, err, msg)
    write(*,*) err, msg
endif
```

NOTES: At each time, the tool determines if the Earth point at (latitude, longitude, altitude) is in the FOV, setting inFOVflag = PGS_TRUE if so, else PGS_FALSE. The vector from SC to Earth point is also returned, whether or not the Earth point is in the FOV, and even if it is on the far side of the Earth. Test for the spacecraft to Earth point being equal to 1.0e50 to avoid processing Earth points that could not be determined because of one or more errors in the transformation.

The FOV is always specified and fixed in SC coordinates. numFOVperimVec should be at least 3. The tool determines if the Earth point lies within the perimeter defined by the vectors perim–FOVvectors[][3]. The first index in C (last in FORTRAN) runs around the perimeter and must be sequential. If the altitude is unknown use zero.

The vector inFOVvector[3] must be defined in SC coordinates and must lie within the FOV. It is necessary for the user to supply a vector within the FOV because on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside the FOV, which outside. If the vector is well centered in the FOV, the algorithm will be faster.

The vectors "perimFOV_vectors[][3]" defining the FOV perimeter can be in clock or counter–clockwise sequence. If the FOV perimeter vectors are supplied out of order, the algorithm will run but the results are unpredictable. The input vectors need not be normalized but must not be zero.

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1090

Get Times of Earth Point in Field of View

NAME: **PGS_CSC_Earthpt_FOV( )**

SYNOPSIS:

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CSC_Earthpt_FOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        char                *earthEllipsTag,
        PGSt_double         latitude,
        PGSt_double         longitude,
        PGSt_double         altitude,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[][3],
        void                *perimFOV_vectors,
        PGSt_boolean        inFOVflag[],
        PGSt_double         sctoEarthptVec[][3])
```

FORTRAN:

```
include'PGS_TD_3.f'
include'PGS_CSC_4.f'
include'PGS_EPH_5.f'
include'PGS_MEM_7.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function pgs_csc_earthpt_fov(numvalues,asciiutc,offsets,
                                     spacecrafttag,earthellipstag,latitude,
                                     longitude,altitude,numfovperimvec,
                                     infovvector,perimfov_vectors,
                                     infovflag,sctoearthptvec)

        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             spacecrafttag
        character*49        earthellipstag
```

```
double precision    latitude
double precision    longitude
double precision    altitude
integer             numfovperimvec
double precision    infovvector(3,*)
double precision    perimfov_vectors(3,*,*)
integer             infovflag(*)
double precision    sctoearthptvec(3,*)
```

DESCRIPTION: For each time value, the tool, using the FOV description, returns a flag or flags indicating if the Earth point of given latitude, longitude and altitude is in the FOV, and a unit vector to that point from the SC in SC coordinates.

INPUTS:

Table 6-214. PGS_CSC_Earthpt_FOV Inputs

Name	Description	Units	Min	Max
numValues	number of time gridpoints	N/A	0	any
asciiUTC	UTC start time	N/A	1972-01-01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values	
spacecraftTag	unique spacecraft identifier	N/A	N/A	N/A
earthEllipsTag	Earth model used	N/A	N/A	N/A
latitude	latitude of Earth point	radians	-pi/2	+pi/2
longitude	longitude of Earth point	radians	-2*pi	+2*pi
altitude	altitude of Earth point	meters	-50000	100000
numFOVperimVec	number of vectors defining FOV perimeter	N/A	3	any
inFOVvector	vector in FOV—preferably near the center in SC coordinates	N/A	N/A	N/A
perimFOV_vectors	vectors in SC coords defining FOV's; MUST be sequential around FOV; the middle dimension must be exactly the same as numFOVperimVec because of the way the array dimensioning works in the function	N/A	N/A	N/A

OUTPUTS:

Table 6-215. PGS_CSC_Earthpt_FOV Outputs

Name	Description	Units	Min	Max
inFOVflag	PGS_TRUE if Earth point is in FOV—see notes	n/a	n/a	n/a
sctoEarthptVec	vector to Earth point in SC coords—returned normalized	meters	-1	1

RETURNS:

Table 6-216. PGS_CSC_Earthpt_FOV Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_BELOW_SURFACE	Location is below surface
PGSCSC_W_BAD_TRANSFORM_VALUE	One or more values in transformation could not be determined
PGSTD_E_BAD_INITIAL_TIME	Initial time is incorrect
PGSTD_E_NO_LEAP_SECS	No leap seconds correction available for input time
PGSCSC_W_DEFAULT_EARTH_MODEL	The default Earth model is used because a correct one was not specified
PGSCSC_W_DATA_FILE_MISSING	The data file earthfigure.dat is missing
PGSCSC_W_SPHERICAL_BODY	Using a spherical Earth model
PGSCSC_W_PROLATE_BODY	Using a prolate Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater then 0.01
PGSCSC_E_INVALID_ALTITUDE	An invalid altitude was specified
PGSCSC_E_NEG_OR_ZERO_RAD	The equatorial or polar radius is negative or zero
PGSMEM_E_NO_MEMORY	No memory is available to allocate vectors
PGSCSC_E_BAD_ARRAY_SIZE	Incorrect array size
PGSCSC_W_PREDICTED_UT1	Status of UT1–UTC correction is predicted
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input times
PGSCSC_E_INVALID_FOV_DATA	FOV perimeter vectors are invalid
PGSCSC_E_FOV_TOO_LARGE	FOV specification outside algorithmic limits
PGSCSC_E_INVALID_EARTH_PT	One of the Earth point vectors was zero
PGSCSC_W_ZERO_PIXEL_VECTOR	Instrument pixel vector of zero length
PGSCSC_W_BAD_EPH_FOR_PIXEL	Ephemeris Data missing for some pixels

EXAMPLES:

C:

```
#define   ARRAY_SIZE   3
#define   PERIMVEC_SIZE 4
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
```

```
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_integer      numValues;
PGSt_double       latitude;
PGSt_double       longitude;
PGSt_double       altitude;
PGSt_integer      numFOVperimVec;
PGSt_double       inFOVvector[ARRAY_SIZE][3] =
                        { {0.0,0.0,100.0},
                          {0.0,0.0,200.0},
                          {0.0,0.0,300.0}
                        };
PGSt_double
      perimFOV_vectors[ARRAY_SIZE][PERIMVEC_SIZE][3]=
                        { {100.0,100.0,100.0},
                          {-100.0,100.0,100.0},
                          {-100.0,-100.0,100.0},
                          {100.0,-100.0,100.0},
                          {200.0,200.0,200.0},
                          {-200.0,200.0,200.0},
                          {-200.0,-200.0,200.0},
                          {200.0,-200.0,200.0},
                          {300.0,200.0,200.0},
                          {-200.0,300.0,200.0},
                          {-200.0,-300.0,300.0},
                          {300.0,-200.0,200.0},
                        };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       sctoEarthptVec[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");
altitude = 10000.0;
latitude = 0.32;
longitude = 2.333;
returnStatus =
PGS_CSC_Earthpt_FOV(numValues,asciiUTC,offsets,
                    PGSd_TRMM,"WGS84",latitude,
                    longitude,altitude,numFOVperimVec,
                    inFOVvector,perimFOV_vectors,inFOVflag,
                    sctoEarthptVec)
if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
```

```
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_earthpt_fov
integer           returnstatus
integer           numvalues
character*27      startutc
double precision  offsets(3)
double precision  latitude
double precision  longitude
double precision  altitude
integer           numfovperimvec
double precision  infovvector(3,4)
double precision  perimfov_vectors(3,4,3)
integer           infovflag(3)
double precision  sctoearthptvec(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

perimfov_vectors(1,1,1) = 100.0
perimfov_vectors(2,1,1) = 100.0
perimfov_vectors(3,1,1) = 100.0

perimfov_vectors(1,2,1) = -100.0
perimfov_vectors(2,2,1) = 100.0
perimfov_vectors(3,2,1) = 100.0

perimfov_vectors(1,3,1) = -100.0
perimfov_vectors(2,3,1) = -100.0
perimfov_vectors(3,3,1) = 100.0

perimfov_vectors(1,4,1) = 100.0
perimfov_vectors(2,4,1) = -100.0
perimfov_vectors(3,4,1) = 100.0

perimfov_vectors(1,1,2) = 200.0
perimfov_vectors(2,1,2) = 200.0
perimfov_vectors(3,1,2) = 200.0

perimfov_vectors(1,2,2) = -200.0
perimfov_vectors(2,2,2) = 200.0
perimfov_vectors(3,2,2) = 200.0
```

```
perimfov_vectors(1,3,2) = -200.0
perimfov_vectors(2,3,2) = -200.0
perimfov_vectors(3,3,2) = 200.0

perimfov_vectors(1,4,2) = 200.0
perimfov_vectors(2,4,2) = -200.0
perimfov_vectors(3,4,2) = 200.0

perimfov_vectors(1,1,3) = 300.0
perimfov_vectors(2,1,3) = 300.0
perimfov_vectors(3,1,3) = 300.0

perimfov_vectors(1,2,3) = -300.0
perimfov_vectors(2,2,3) = 300.0
perimfov_vectors(3,2,3) = 300.0

perimfov_vectors(1,3,3) = -300.0
perimfov_vectors(2,3,3) = -300.0
perimfov_vectors(3,3,3) = 300.0

perimfov_vectors(1,4,3) = 300.0
perimfov_vectors(2,4,3) = -300.0
perimfov_vectors(3,4,3) = 300.0

infovvector(1,1) = 0.0
infovvector(1,2) = 0.0
infovvector(1,3) = 100.0

infovvector(2,1) = 0.0
infovvector(2,2) = 0.0
infovvector(2,3) = 200.0

infovvector(3,1) = 0.0
infovvector(3,2) = 0.0
infovvector(3,3) = 300.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4
altitude = 10000.0
latitude = 0.32
longitude = 2.333

returnstatus =
pgs_csc_earthpt_fov(numvalues,startutc,offsets,
                    PGSd_TRMM,'WGS84',latitude,
                    longitude,altitude,numfovperimvec,
                    infovvector,perimfov_vectors,
                    infovflag,sctoearthptvec)
```

```
if (returnstatus .ne. pgs_s_success) then
    pgs_smf_getmsg(returnstatus, err, msg)
    write(*,*) err, msg
endif
```

NOTES: At each time, the tool determines if the Earth point at (latitude, longitude, altitude) is in the FOV, setting inFOVflag = PGS_TRUE if so, else PGS_FALSE. The vector from SC to Earth point is also returned, whether or not the Earth point is in the FOV, and even if it is on the far side of the Earth. Test for the spacecraft to Earth point being equal to 1.0e50 to avoid processing Earth points that could not be determined because of one or more errors in the transformation.

The FOV is always specified in SC coordinates. For an instrument fixed to the SC, use the same FOV description always. For scanning instruments, user should provide the description appropriate to the scan instrument. numFOVperimVec should be at least 3. The tool determines if the Earth point lies within the perimeter defined by the vectors perim–FOVvectors[][][3]. The first index in C (last in FORTRAN) is the time offset index and the second must be sequential around the FOV perimeter. If the altitude is unknown use zero.

The vector inFOVvector[][3] must be defined in SC coordinates and must lie within the FOV. The last index in C, (first in FORTRAN) on these vectors is for X,Y, and Z, components in SC coordinates. It is necessary for the user to supply a vector within the FOV because on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside the FOV, which outside. If the vector is well centered in the FOV, the algorithm will be faster.

The vectors "perimFOV_vectors[][][3]" defining the FOV perimeter can be in clock or counter–clockwise sequence. If the FOV perimeter vectors are supplied out of order, the algorithm will run but the results are unpredictable. The input vectors need not be normalized but must not be zero.

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REQUIREMENTS: PGSTK–1090

Estimate Refraction of Ray

NAME: **PGS_CSC_SpaceRefract( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SpaceRefract(
        PGSt_double  spaceZenith,
        PGSt_double  altitude,
        PGSt_double  latitude,
        PGSt_double  *surfaceZenith,
        PGSt_double  *displacement)
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_spacerefract(spacezenith,altitude,
                                      latitude,surfacezenith, displacement)
        double precision    spacezenith
        double precision    altitude
        double precision    latitude
        double precision    surfacezenith
        double precision    displacement
```

DESCRIPTION: This function estimates the refraction of a ray incident from space or a line of sight from space to the Earth's surface based on the unrefracted zenith angle (most common algorithms, intended for ground based observation, require knowledge of the refracted, not the unrefracted zenith angle). The algorithm is suitable for:

a. approximate determination of the apparent Solar zenith angle from the true (geometrical, unrefracted) Solar zenith angle (obviously, also applicable to Lunar zenith angle, etc.)

b. correction of the viewing angle from space, to approximately remove the effects of refraction

The method is briefly indicated in the NOTES, q.v. for various caveats.

INPUTS:

Table 6-217. PGS_CSC_SpaceRefract Inputs

Name	Description	Units	Min	Max
spaceZenith	unrefracted zenith angle	radians	0	pi/2 (90 deg)
altitude	altitude off the geoid	meters	-1000	50000
latitude	latitude	radians	-pi/2	pi/2

OUTPUTS:

Table 6-218. PGS_CSC_SpaceRefract Outputs

Name	Description	Units	Min	Max
surfaceZenith	refracted zenith angle	radians	0	n/a
displacement	displacement of the footpoint of ray	radians	0	~0.01

RETURNS:

Table 6-219. PGS_CSC_SpaceRefract Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGS_CSC_BAD_LAT	a latitude out of the range (- pi/2, pi/2) was entered
PGS_CSC_E_INVALID_ZENITH	a negative zenith angle was entered
PGSCSC_W_INVALID_ALTITUDE	Attempt to calculate refraction at point too far below Earth's surface
PGSCSC_W_BELOW_HORIZON	Attempt to calculate refraction of ray below horizon

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double spaceZenith=0.4;
PGSt_double altitude=5000.0;
PGSt_double latitude= - 0.2 ;  **** not implemented at
                                    present ***
PGSt_double surfaceZenith;
PGSt_double displacement;

returnStatus = PGS_CSC_SpaceRefract(spaceZenith,altitude,
                                    latitude,&surfaceZenith,
                                    &displacement)

{
  ** test errors,
      take appropriate
```

```
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_spacerefract
integer           returnstatus
double precision  spacezenith
double precision  altitude
double precision  latitude
double precision  surfacezenith
double precision  displacement

data spacezenith /0.4/
data altitude /5000.0/
data latitude /-0.2/

returnstatus = pgs_csc_spacerefract(spacezenith,altitude,
                                    latitude,surfacezenith,
                                    displacement)

if(returnstatus .ne. pgs_s_success) go to 90
write(6,*) surfacezenith,displacement

90  write(6,99)returnstatus
99  format('ERROR:',I15)
```

NOTES:

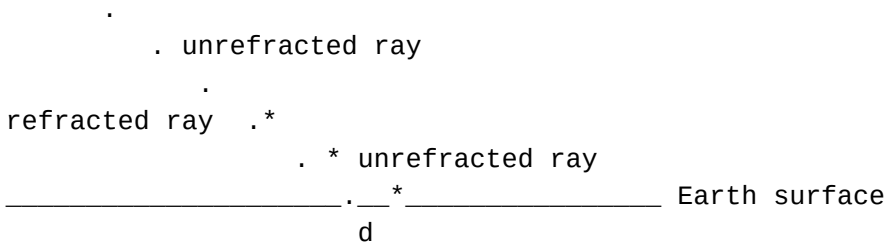
This algorithm is intended as a mean-atmosphere approximation, valid for white light (for example, sunlight). Refraction is quite wavelength dependent, and in the atmosphere it will also depend strongly on local conditions (e.g., the weather). The present algorithm is intended to be a reasonable approximation such that to do better one would need local and, for large zenith angles, regional weather.

Caveat: The altitude is used ONLY to obtain the air pressure, which is then used to obtain the surface index of refraction. Users who employ an inflated Earth radius in geolocation should be especially careful to replace any derived altitude with the height in meters above the geoid before calling this function.

The method is based on the author's calculations, using a conservation law originally due to W. Chauvenet, for the important difference z0 - z', and an empirical refraction algorithm in Equation 3.283-1, p. 144, Astron. Almanac Supplement (U.S. Naval Observatory) to derive the less important displacement.

The (horizontal) displacement of the ray is in a vertical plane containing the ray and is in the sense that the actual (refracted) ray will meet the Earth d = (displacement)*Re meters from the geometrical (unrefracted) position, on the side towards the horizon.

```
 Outer Space  Here
```



the angle "displacement" is the angle that the displacement in meters "d" subtends at Earth center.

The following table exemplifies results at sea level, using a conversion of 6371000 m per radian on the displacement.

Table 6-220. Altitude – Sea Level

Zenith Angle in Space (deg)	Zenith Angle at Surface (deg)	Refraction (deg)	Linear Displacement (meters)
10.000000	9.997066	0.002934	0.549064
20.000000	19.993944	0.006056	1.222937
30.000000	29.990394	0.009606	2.221314
40.000000	39.986039	0.013961	3.982978
45.000000	44.983363	0.016637	5.464087
50.000000	49.980174	0.019826	7.725334
55.000000	54.976243	0.023757	11.398788
60.000000	59.971192	0.028808	17.845724
61.000000	60.969996	0.030004	19.696711
62.000000	61.968722	0.031278	21.816620
63.000000	62.967361	0.032639	24.256691
64.000000	63.965905	0.034095	27.080360
65.000000	64.964340	0.035660	30.366779
70.000000	69.954333	0.045667	58.380584
75.000000	74.938025	0.061975	136.072953
76.000000	75.933417	0.066583	166.728721
77.000000	76.928121	0.071879	207.384912
78.000000	77.921967	0.078033	262.469333
79.000000	78.914723	0.085277	338.977167
80.000000	79.906069	0.093931	448.379942
81.000000	80.895543	0.104457	610.332976
82.000000	81.882461	0.117539	860.316290
83.000000	82.865762	0.134238	1266.536004
84.000000	83.843713	0.156287	1970.638000
85.000000	84.813286	0.186714	2974.066487
86.000000	85.768718	0.231282	4858.394025
87.000000	86.697712	0.302288	8677.416632
88.000000	87.569758	0.430242	17538.457911
89.000000	88.295108	0.704892	41818.325388
90.000000	88.619113	1.380887	113429.256196

Note that the linear displacement at 88 degrees zenith angle is about 17.5 km—very substantial. Because of the very approximate atmosphere model,

this number could vary by perhaps 25% depending on weather in temperate and tropical regions; in the Arctic it would be considerably smaller. The displacement at 90 degrees incidence, over 113 km, is only suggestive and could easily vary by 50%.

The increments in latitude and longitude due to refraction are:

Direction	**Value**
latitude (ϕ)	dAng * cos(ψ)
longitude (λ)	dAng * sin(ψ)/cos(ϕ)

where ψ is the azimuth from PGS_CSC_ZenithAzimuth(). The expression for longitude is singular at the North and South poles and the user should avoid using it there, or within too close range. When |latitude| > π - dAng, the point is so near the pole that the displacement of the ray can be assumed to be South at the North pole and North at the South pole; but when starting at either pole, the longitude (not its increment) must be found from -atan2(yray,xray) where (xray,yray,zray) are the components of the look vector in ECR. After calling PGS_CSC_SpaceRefract(), then, the user who is interested in the displacement in latitude and longitude needs to implement the equations above and, for the exceptional case at a pole, the alternate just explained: latitude = dAng, longitude = -atan2(yray,xray). The Toolkit software does not perform these operations, which are a user responsibility if the positional correction is desired.

The composition of the atmosphere was obtained from Allen's "Astrophysical Quantities, 2nd ed." (London, the Athlone Pre, 1976) p. 121, because the U.S. Standard Atmosphere (NOAA, 1976) is bone dry, which is unrealistic.

The atmosphere model is used only to get the index of refraction at sea level. The latitude dependence is that the sea level temperature and mean scale height are functions of latitude.

The calculations are based on the geometry of a spherical Earth. User may employ her/his favorite Earth radius to transform radians of displacement to meters. See also “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger, where the equation to transform displacement magnitude to North and East components is given.

REQUIREMENTS: PGSTK–0860, PGSTK-1080

Get Field-of-View Footprint and Pixel Centers

NAME: **PGS_CSC_GetFOV_Pixel( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GetFOV_Pixel(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        char                earthEllipsTag[50],
        PGSt_boolean        accurFlag,
        PGSt_double         pixelUnitvSC[][3],
        PGSt_double         offsetXYZ[][3],
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_double         pixelUnitvECR[][3],
        PGSt_double         slantRange[],
        PGSt_double         velocDoppl[])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_getfov_pixel(spacecrafttag,numvalues,asciiutc,
                                      offsets, earthellipstag,accurflag,
                                      pixelunitvsc,offsetxyz,latitude,
                                      longitude,pixelunitvecr,
                                      slantrange,velocdoppl)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        character*49        earthellipstag
        integer             accurflag
        double precision    pixelunitvsc(3,*)
        double precision    offsetxyz(3,*)
        double precision    latitude(*)
```

```
double precision     longitude(*)
double precision     pixelunitvecr(3,*)
double precision     slantrange(*)
double precision     velocdoppl(*)
```

DESCRIPTION: This function obtains the latitude and longitude of the intersection of a line of sight with the spheroidal Earth, the slant range from Spacecraft to look point, and the Doppler velocity along the line of sight. The ECR pixel vector is also returned; it can be used, for example, to determine the zenith angle of the line of sight. The line of sight is defined by a unit vector in the Spacecraft frame of reference and a time. (The unit vector along the line of sight is called a "look vector" in the sequel.)

The Doppler velocity is true, in the sense that it is relative to the Earth's surface.

INPUTS:

Table 6-221. PGS_CSC_GetFOV_Pixel Inputs

Name	Description	Units	Min	Max
spacecraftTag	spacecraft identifier	N/A	N/A	N/A
numValues	number of input time offsets (to use ASCII time with no offsets, set numValues =0 or set it =1 and make first [and only] offset = 0.0)	N/A	0	N/A
asciiUTC	UTC start time in CCSDS ASCII Time A or B format	N/A	1972-01-01	see NOTES
offsets	array of time offsets	SI seconds	Max and Min such that floating equivalent of asciiUTC+offset is between asciiUTC Min and Max values	
EarthEllipsTag	tag selecting Earth Ellipsoid model	N/A	N/A	N/A
accurFlag	flag to regulate accuracy	N/A	PGS_FALSE	PGS_TRUE
pixelUnitvSC	array of pixel unit vectors in SC coords	N/A	-1	1
offsetXYZ	array of displacements of instrument boresight from SC nominal center in SC coordinates(see overall limit for length of this vector in "RETURNS" section) (offsetXYZ is used only when accurFlag == PGS_TRUE)	m	-120	+120

OUTPUTS:

Table 6-222. PGS_CSC_GetFOV_Pixel Outputs

Name	Description	Units	Min	Max
latitude	latitude of the lookpoint	radians	-pi/2	pi/2
longitude	longitude of the lookpoint	radians	-pi	pi
pixelUnitvECR	ECR unit pixel vector	N/A	-1	+1
slantRange	slant range: SC to lookpoint	m	0	100000
velocDoppl	Doppler velocity of the look point (+ meaning "away")	m/s	-8000	8000

RETURNS:

Table 6-223. PGS_CSC_GetFOV_Pixel Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_W_MISS_EARTH	Look Vector fails to intersect Earth
PGSTD_E_SC_TAG_UNKNOWN	Invalid Spacecraft tag
PGSCSC_W_ZERO_PIXEL_VECTOR	Instrument pixel vector of zero length
PGSCSC_W_BAD_EPH_FOR_PIXEL	Ephemeris Data missing for some pixels
PGSCSC_W_INSTRUMENT_OFF_BOARD	Instrument offset from SC center is > 120 m which is considered unreasonably large (applicable only when accurFlag = PGS_TRUE)
PGSCSC_W_BAD_ACCURACY_FLAG	Accuracy Flag neither PGS_TRUE nor PGS_FALSE
PGSCSC_E_BAD_ARRAY_SIZE	The user has supplied a negative number of time offsets
PGSCSC_W_DEFAULT_EARTH_MODEL	Invalid EarthEllipsTag; WGS84 model used
PGSCSC_W_DATA_FILE_MISSING	A file such as the ephemeris, utcpole, Earth Model or leap seconds file is missing
PGSCSC_E_NEG_OR_ZERO_RAD	One of the Earth axes is zero or negative
PGSMEM_E_NO_MEMORY	Malloc operation for scratch memory failed
PGSTD_E_NO_LEAP_SECS	no leap seconds data available for input time
PGSTD_E_TIME_FMT_ERROR	format error in asciiUTC
PGSTD_E_TIME_VALUE_ERROR	value error in asciiUTC
PGSCSC_W_PREDICTED_UT1	predicted UT1 value used
PGSCSC_E_NO_UT1_VALUE	no UT1 value available
PGS_E_TOOLKIT	Error in Toolkit—for example, inconsistent error message from a subordinate function
PGSEPH_E_BAD_EPHEM_FILE_HDR	No s/c ephem files had readable headers
PGSEPH_E_NO_SC_EPHEM_FILE	No s/c ephem files could be found for input

EXAMPLES:

C:

```
#include <PGS_CSC.h>
char        asciiUTC[28] = "1994-01-15T12:21:33.9939Z";
PGSt_tag          spacecraftTag = PGSd_EOS_AM;
char              EarthEllipsTag[50] = "WGS84";
PGSt_double       offsets[4]  = {0.0,0.1,2.0,30.0};
PGSt_double       pixelUnitvSC[4][3];
PGSt_double       offsetXYZ[4][3];
PGSt_integer      numValues = 4;
PGSt_boolean      accurFlag = PGS_FALSE;

PGSt_double       latitude[4];
PGSt_double       longitude[4];
PGSt_double       velocDoppl[4];
PGSt_double       slantRange[4];
PGSt_double       pixelUnitvECR[4][3];
PGSt_SMF_status   returnStat;
```

```
PGSt_SMF_status   code;
char              msg[240];
char              mnemonic[31];
int               i;
int               jj;

for (i=0;i<4;i++)
     for(jj=0;jj<3 ;++jj)
         offsetXYZ[i][jj] = 0.0;

/** initialize pixel unit vectors
   All but the 3rd case hit Earth; to miss Earth reverse
   the last component of any other one **/

pixelUnitvSC[0][0] = 0.03;
pixelUnitvSC[0][1] = 0.12;
pixelUnitvSC[0][2] = 0.08;

pixelUnitvSC[1][0] =  -0.2;
pixelUnitvSC[1][1] =  0.12;
pixelUnitvSC[1][2] =  0.6;

/**This case will display error**/

pixelUnitvSC[2][0] =  -0.0;
pixelUnitvSC[2][1] =  0.00;
pixelUnitvSC[2][2] =  0.0;

pixelUnitvSC[3][0] =  -0.2;
pixelUnitvSC[3][1] =  -0.12;
PixelUnitvSC[3][2] =  0.6;

returnStat = PGS_CSC_GetFOV_Pixel(spacecraftTag,numValues,
                                  asciiUTC,offsets,
                                  EarthEllipsTag,accurFlag,
                                  pixelUnitvSC,offsetXYZ,
                                  latitude,longitude,
                                  pixelUnitvECR,santRange,
                                  velocDoppl);

printf("  Toolkit return value:  %d\n\n",returnStat);

PGS_SMF_GetMsg(&code,mnemonic,msg);
printf("  Return %s: %s\n\n",mnemonic,msg);

printf(" accurFlag ==  %d  Earth Tag == %s  ECR Pixels:\n"
        "%15.11lg    %15.11lg    %15.11lg\n"
        "%15.11lg    %15.11lg    %15.11lg\n "
        "%15.11lg    %15.11lg    %15.11lg\n"
```

```
        "%15.11lg    %15.11lg    %15.11lg\n",
        accurFlag,EarthEllipsTag,
        pixelUnitvECR[0][0],pixelUnitvECR[0][1],
            pixelUnitvECR[0][2],
        pixelUnitvECR[1][0],pixelUnitvECR[1][1],
            pixelUnitvECR[1][2],
        pixelUnitvECR[2][0],pixelUnitvECR[2][1],
            pixelUnitvECR[2][2],
        pixelUnitvECR[3][0],pixelUnitvECR[3][1],
            pixelUnitvECR[3][2]);

/**  Test for some variable like latitude =
    PGSd_GEO_ERROR_VALUE before further processing to avoid
    processing pixels that missed Earth or had zero pixel
    vector. In multi-pixel processing, results from good and
    bad pixels can be distinguished only by answers being
    PGSd_GEO_ERROR_VALUE; in single pixel processing return
    status indicates any error **/

if(returnStatus != PGS_S_SUCCESS)
{
  /**  print results - latitude, longitude, etc.; test
      errors, take appropriate action **/
}
```

FORTRAN:

```
implicit none

parameter(numPixels = 4)
integer           pgs_csc_getfov_pixel
integer           spacecrafttag
integer           numvalues
character*27      asciiutc
double precision  offsets(numPixels)
character*49      earthellipstag
integer           accurflag
double precision  pixelUnitvSC(3,numPixels)
double precision  offsetXYZ(3,numPixels)
double precision  latitude(numPixels)
double precision  longitude(numPixels)
double precision  pixelUnitvECR(3,numPixels)
double precision  slantRange(numPixels)
double precision  velocDoppl(numPixels)
character*33      err
character*241     msg
```

```
      data offsets/360.0, 720.0, 1080.0, 1600.0/
      asciiutc = '1991-07-27T11:04:57.987654Z'
      spacecrafttag = PGSd_EOS_AM

      do 1 jj = 1,3
      do 1 i = 1,4
         offsetXYZ(jj,i) = 0.0;

1      continue

!

!     This puts instrument at the nominal SC center

1     For example, to put instrument on a 20 m boom fore of

!     SC center, make  offsetXYZ(1,i) = 20.0 for each i

!    initialize pixel unit vectors

!    All but the 3rd case hit Earth; to miss Earth reverse the

!    last component of any other one

      pixelUnitvSC(1,1) = 0.03;
      pixelUnitvSC(2,1) = 0.12;
      pixelUnitvSC(3,1) = 0.08;

      pixelUnitvSC(1,2) =  -0.2;
      pixelUnitvSC(2,2) =  0.12;
      pixelUnitvSC(3,2) =  0.6;

!      This case will display error

      pixelUnitvSC(1,3) =  -0.0;
      pixelUnitvSC(2,3) =  0.00;
      pixelUnitvSC(3,3) =  0.0;

      pixelUnitvSC(1,4) =  -0.2;
      pixelUnitvSC(2,4) =  -0.12;
      pixelUnitvSC(3,4) =  0.6;

      returnstatus = pgs_csc_getfov_pixel(spacecrafttag,numvalues,
      >                                   asciiutc,offsets,
      >                                   earthellipstag,
      >                                   accurflag,pixelUnitvSC,
      >                                   offsetXYZ,latitude,
      >                                   longitude,pixelUnitvECR,
      >                                   slantRange,velocDoppl)

!    Print output values
```

```
!               Test for some variable like latitude = 1.0e50 before further

!               processing to avoid processing pixels that missed Earth or had

!               zero pixel vector

                   if (returnstatus .ne. pgs_s_success) then
                       pgs_smf_getmsg(returnstatus, err, msg)
                       write(*,*) err, msg
                   endif
```

NOTES: An accuracy flag is required, allowing two accuracy levels:

Normal or PGS_FALSE

- do ECI to ECR transformation at moment of taking data
- consider instrument axis to pass through nominal center of spacecraft

High or PGS_TRUE

- do ECI to ECR transformation with approximate allowance for Earth rotation during the light travel time (spherical Earth approximation.) This will slow the calculation slightly.
- user must supply vector offsetXYZ that represents the displacement in meters of the instrument boresight from nominal spacecraft center. (Only the part of the displacement orthogonal to the look vector will have an effect.) Users invoking the High Accuracy option but wishing not to take advantage of this feature should supply zeros for the components of offsetXYZ.

The maximum error in omitting this calculation is approximately as follows for a worst case of a spacecraft at 700 km altitude, crossing the equator and looking E or W:

Table 6-224. Error due to Earth Motion in Time of Flight of Light

Nadir Angle (deg)	Slant Range (km)	Worst Case Error (m) if accurFlag = PGS_FALSE
0	700	1.1
30	830	1.3
40	945	1.5
50	1200	1.9
55	1410	2.1
60	1770	2.7
64	2440	3.7

The nature of the error is a smooth distortion such that points near either the East or the West limb would be assigned a longitude slightly to the West in comparison with points near nadir. The effect could be somewhat exaggerated, for some orbits, in terms of illumination changes near the terminator.

Caution: The user is advised that the spacecraft ephemeris refers to the nominal center of the spacecraft. The displacements of individual instruments relative to the center of the spacecraft are taken into account herein through the vector offsetXYZ. When the flag "accurFlag" is set to PGS_TRUE, the user should specify the instrument coordinates relative to spacecraft center (in meters) with this vector. It WILL be used by the present function, so if the user does not actually wish to employ it, then offsetXYZ must be set to zero (all three components). If "accurFlag" is set to PGS_FALSE, the displacement is ignored.

TIME ACRONYMS:

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

REFERENCES FOR TIME:

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. See also “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

REQUIREMENTS: PGSTK–0930, PGSTK–1080, PGSTK–1083,

Precesses a Vector Between TDB Julian Date and J2000 Coordinates

NAME: **PGS_CSC_precs2000( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_precs2000(
        PGSt_integer        threeOr6,
        PGSt_double         jedTDB[2],
        PGSt_boolean        frwd,
        PGst_double         posVel[])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD.4.f'

integer function pgs_csc_precs2000(threeor6,jedtdb,fwrd,posvel)
        integer             threeor6
        double precision    jedtdb(2)
        integer             frwd
        double precision    posvel(6)
```

DESCRIPTION: This tool precesses 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).

INPUTS:

Table 6-225. PGS_CSC_precs2000 Inputs

Name	Description	Units	Min	Max
jedTDB[2]	TBD (Barycentric Dynamical Time) as a Julian Date to or from which the vector is to be processed	days	ANY	ANY
frwd	flag for sense of precession: PGS_TRUE if precessing from J2000 to jedTDB PGS_FALSE if precessing from jedTDB to J2000	T/F	N/A	N/A
posVel	vector (position and velocity) in final reference frame: posvel[0-2] position posvel[3-5] velocity	 m m/s	 ANY ANY	 ANY ANY

OUTPUTS:

Table 6-226. PGS_CSC_precs2000 Outputs

Name	Description	Units	Min	Max
posVel	vector (position and velocity) in final reference frame:			
	posvel[0-2] position	m	ANY	ANY
	posvel[3-5] velocity	m/s	ANY	ANY

RETURNS:

Table 6-227. PGS_CSC_precs2000 Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_BAD_ARRAY_SIZE	The size of the vector is not either 3 or 6
PGSCSC_E_BAD_DIRECTION_FLAG	The value of the direction flag is not either PGS_TRUE or PGS_FALSE

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       posVel[6]={6400000.0,-5000000.0,40000.0,
                             4000.0,7000.0,-6000.0};

** precess the vector **

returnStatus = PGS_CSC_precs2000(6,jedTDB,PGS_TRUE,posVel);

** the input vector "posVel" has been overwritten with the
   precessed value **
```

FORTRAN:

```
implicit none

integer           pgs_csc_precs2000
integer           returnstatus
integer           threeor6
double precision  jedtdb(2)
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
```

```
                returnstatus = pgs_csc_nutate2000(threeor6,jedtdb,frwd,
                                                  posvel)

!      the input vector "posvel" has been overwritten with the precessed value
```

NOTES:

This function is a simplified version of 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.

TIME ACRONYMS:

TDB is: Barycentric Dynamical Time

JULIAN DATES:

Format:

Toolkit Julian dates are kept as an array of two real (high precision) numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and return a Julian date will return the Julian date in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert the input date (internal) to the above format. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but will not check and the format of the output may not be what is expected if any other format is used for the input.

Meaning:

Toolkit "Julian dates" are all based on UTC. A Julian date in any other "time" (e.g., TAI, TDT, UT1, etc.) is based on the difference between that "time" and the equivalent UTC time (differences range in magnitude from 0 seconds to about a minute).

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK–0930, PGSTK–1050

Nutate State Vector Between True of Date and Mean of Date

NAME: **PGS_CSC_nutate2000( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_nutate2000(
        PGSt_integer        threeOr6,
        PGSt_double         jedTDB[2],
        PGSt_double         dvnut[4],
        PGSt_boolean        frwd,
        PGst_double         posVel[])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'

integer function pgs_csc_nutate2000(threeor6,jedtdb,dvnutfwrd,posvel)
        integer             threeor6
        double precision    jedtdb(2)
        double precision    dvnut(4)
        double precision    frwd
        double precision    posvel(*)
```

DESCRIPTION: This tool transforms 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.

INPUTS:

Table 6-228. PGS_CSC_nutate2000 Inputs (1 of 2)

Name	Description	Units	Min	Max
threeOr6	chooses a 3 or 6 dimensional vector to nutate	N/A	N/A	N/A
jedTDB	TBD (Barycentric Dynamical Time) as a Julian Date to or from which the vector is to be nutated (this variable is generally referred to in Toolkit code as jedTDB	days	ANY	ANY
dvnut	the two nutation angles and their rates, output from “PGS_CSC_wahr2” (this variable is generally referred to in Toolkit code as dvnut)	rad/s	-1.e-11	1.e-11
posVel	vector (position and velocity) in initial reference frame: posvel[0-2] position posvel[3-5] velocity	 m m/s	 ANY ANY	 ANY ANY

Table 6-228. PGS_CSC_nutate2000 Inputs (2 of 2)

Name	Description	Units	Min	Max
frwd	flag for sense of nutation: PGS_TRUE if nutating fromTrue of Date at jedTDB to Mean of Date PGS_FALSE if nutating from Mean of Date to True of Date at jedTDB	T/F	N/A	N/A

OUTPUTS:

Table 6-229. PGS_CSC_nutate2000 Outputs

Name	Description	Units	Min	Max
posVel	vector (position and velocity) in final reference frame:			
	posvel[0-2] position	m	ANY	ANY
	posvel[3-5] velocity	m/s	ANY	ANY

RETURNS:

Table 6-230. PGS_CSC_nutate2000 Returns

Return	Description
PGS_S_SUCCESS	Success
PGSCSC_E_BAD_ARRAY_SIZE	The size of the vector is not either 3 or 6
PGSCSC_E_BAD_DIRECTION_FLAG	The value of the direction flag is not either PGS_TRUE or PGS_FALSE

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       dvnut[4];
PGSt_double       posVel[6]={6400000.0,-5000000.0,40000.0,
                             4000.0,7000.0,-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 **
```

FORTRAN:

```
      implicit none

      integer           pgs_csc_nutate2000
      integer           returnstatus
      integer           threeor6
      double precision  jedtdb(2)
      double precision  dvnut
      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

!     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
```

NOTES:

Purpose: The case of transforming a vector from J2000 to True of Date, requires first procession and then nutation. The intermediate system, processed but not nutated, is the Mean of Date system. With the direction flag at PGS_TRUE, this function transforms a vector (position and velocity) from Mean of Date to True of Date. True of date has its Z axis along the Earth's true angular velocity and the X axis is toward the true equinox of date-the intersection of the equator perpendicular to Z with the ecliptic. Mean of date is arranged similarly, but ignoring nutation, so its pole has a constant angle to the ecliptic, along which its X axis moves at a constant rate.

In the opposite case, with the direction flag at PGS_FALSE, i.e. in going from arbitrary epoch to J2000, this function carries the vector from True of Date to Mean 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.

TIME ACRONYMS:

TDB is: Barycentric Dynamical Time

JULIAN DATES:

Format:

Toolkit Julian dates are kept as an array of two real (high precision) numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and return a Julian date will return the Julian date in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert the input date (internal) to the above format. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but will not check and the format of the output may not be what is expected if any other format is used for the input.

Meaning:

Toolkit "Julian dates" are all based on UTC. A Julian date in any other "time" (e.g., TAI, TDT, UT1, etc.) is based on the difference between that "time" and the equivalent UTC time (differences range in magnitude from 0 seconds to about a minute).

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK–0914, PGSTK–0930, PGSTK–1050

Transform from ECI J2000 to ECI True of Date Coordinates

NAME: **PGS_CSC_J2000toTOD( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_J2000toTOD(
        PGSt_integer        threeOr6,
        PGSt_double         secTAI93
        PGSt_double         posvelECI[6],
        PGSt_double         posvelTOD[6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_SMF.f'

integer function pgs_csc_j2000totod(threeor6,sectai93,posveleci,
>                                   posveltod)
        integer             threeor6
        double precision    sectai93
        double precision    posveleci(*)
        double precision    posveltod(*)
```

DESCRIPTION: This function transforms from ECI (J2000) coordinates to TOD (true of date) coordinates.

INPUTS:

Table 6-231. PGS_CSC_J2000toTOD.c Inputs

Name	Description	Units	Min	Max
threeOr6	dimension of input vector	N/A	3	6
secTAI93	TOD time	seconds		
posvelECI[]	Vector (position and possibly velocity) in ECI J2000			
posvelTOD[0]	x position	meters		
posvelTOD[1]	y position	meters		
posvelTOD[2]	z position	meters		
posvelTOD[3]	x velocity	m/s		
posvelTOD[4]	y velocity	m/s		
posvelTOD[5]	z velocity	m/s		

OUTPUTS:

Table 6-232. PGS_CSC_J2000to.TOD.c Outputs

Name	Description	Units	Min	Max
posvelTOD[6]	Vector (position and possibly velecity) in ECI TOD			
posvelECI[0]	x position	meters		
posvelECI[1]	y position	meters		
posvelECI[2]	z position	meters		
posvelECI[3]	x velocity	m/s		
posvelECI[4]	y velocity	m/s		
posvelECI[5]	z velocity	m/s		

RETURNS:

Table 6-233. PGS_CSC_J2000toTOD Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_E_BAD_ARRAY_SIZE	incorrect array size

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double secTAI93 = -44496000.0;
PGSt_double posvelECI[6] = {0.5,0.75,0.90,0.3,0.2,0.8};
PGSt_double posvelTOD[6];

returnStatus=
PGS_CSC_J2000toTOD(6,secTAI93,posvelECI,posvelTOD);

if(returnStatus != PGS_S_SUCCESS)
{
/** test errors, take appropriate action **/
}
```

FORTRAN:

```
implicit none

integer           returnstatus
integer           pgs_csc_j2000totod
integer           threeor6
double precision  sectai93
double precision  posveleci(6)
```

```
double precision  posveltod(6)
integer           cnt1
character*33      err
character*241     msg

do 10 cnt1 = 1,6
      posveleci(cnt1) = 100 * cnt1
10 continue
sectai93 = -44496000.0
three0r6 = 6

returnstatus=s_csc_j2000totod(threeor6,sectai93,posveleci,
                                  posveltod)

if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err, msg
endif
```

NOTES: If threeOr6 is 3, only position is transformed; if 6 then both position and velocity.

REQUIREMENTS: PGSTK - 0910, 1050

Transform from ECI True of Date to ECI J2000 Coordinates

NAME: **PGS_CSC_TODtoJ2000( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_TODtoJ2000(
        PGSt_integer    threeOr6,
        PGSt_double     secTAI93,
        PGSt_double     posvelTOD[6],
        PGSt_double     posvelECI[6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'

include 'PGS_SMF.f'

integer function pgs_csc_todtoj2000(threeor6,sectai93,posveltod,
                                    posveleci)
        integer
        double precision
        double precision posveltod(*)
        double precision posveleci(*)
```

DESCRIPTION: This function transforms from TOD (true of date) coordinates to ECI (J2000) coordinates.

INPUTS:

Table 6-234. PGS_CSC_TODtoJ2000.c Inputs

Name	Description	Units	Min	Max
threeOr6	dimension of input vector	N/A	3	6
secTAI93	TOD time posvel TOD[6]	seconds		
posvelTOD[]	Vector (position and possibly velocity) in ECI TOD			
posvelTOD[0]	x position	meters		
posvelTOD[1]	y position	meters		
posvelTOD[2]	z position	meters		
posvelTOD[3]	x velocity	m/s		
posvelTOD[4]	y velocity	m/s		
posvelTOD[5]	z velocity	m/s		

OUTPUTS:

Table 6-235. PGS_CSC_TODtoJ2000.c Outputs

Name	Description	Units	Min	Max
posvelECI[]	Vector (position and possibly velecity) in ECI J2000			
posvelECI[0]	x position	meters		
posvelECI[1]	y position	meters		
posvelECI[2]	z position	meters		
posvelECI[3]	x velocity	m/s		
posvelECI[4]	y velocity	m/s		
posvelECI[5]	z velocity	m/s		

RETURNS:

Table 6-236. PGS_CSC_TODtoJ2000c Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_E_BAD_ARRAY_SIZE	incorrect array size

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus
PGSt_double secTAI93 = -44496000.0
PGSt_double posvelTOD[6] = 0.5,0.75,0.90,0.3,0.2,0.8};
PGSt_double posvelECI[6];
returnStatus =
PGS_CSC_TODtoJ2000(6,secTAI93,posvelTOD,posvelECI);
if(returnStatus != PGS_S_SUCCESS
{
/** test errors, take appropriate  action **/
}
```

FORTRAN:

```
implicit none
integer           pgs_csc_todtoj2000
integer           returnstatus
integer           threeor6
double precision  sectai93
double precision  posveltod(6)
double precision  posveleci(6)
integer     cnt1
character*33      err
character*241     msg
```

```
do 10 cnt1  = 1,6
      posveltod(cnt1) = 100 * cnt1
10 continue
sectai93 = -44496000.0
three0r6 = 6
returnstatus=pgs_csc_todtoj2000(threeor6,sectai93,posveltod,
>                                    posveleci)
if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err,
endif
```

NOTES: If threeor6 is 3, only position is transformed; if 6 then both position and velocity.

TIME ACRONYMS:

TAI is: International Atomic Time

TDB is: Barycentric Dynamical Time

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to inthe toolkit as TAI.

REFERENCES FOR TIME:

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

REQUIREMENTS: PGSTK - 0910, 1050

Determine if Location on Earth is in Day or Night

NAME: **PGS_CSC_DayNight( )**

SYNOPSIS:

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CSC_DayNight(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_tag            sunZenithLimitTag,
        PGSt_boolean        afterDark[])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_CBP_6.f'
include 'PGS_CSC_4.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_daynight(numvalues,asciiutc,offsets,latitude,
                                  longitude, sunzenithlimittag, afterdark)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    latitude(*)
        double precision    longitude(*)
        integer             sunzenithlimittag
        integer             afterdark(*)
```

DESCRIPTION: This function determines whether each point in a set of input Earth locations is in day or night at the corresponding input times. The function accepts an input start time, array of offsets from that start time, and an array of corresponding geodetic latitudes and longitudes. It then determines whether each time and point on the surface of the Earth (altitude = 0 km) is night, based on definitions of either civil twilight or night, nautical night, or astronomical night.

INPUTS:

Table 6-237. PGS_CSC_DayNight Inputs

Name	Description	Units	Min	Max
numValues	number of input time offsets, longitudes, and latitudes	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	1972–01-01	see NOTES
offsets	array of time offsets	seconds	Max and Min such that asciiUTC+ offset is between asciiUTC Min and Max values	
latitude	array of geodetic latitudes for array of time offsets	radians	-pi/2	+pi/2
longitude	array of longitudes corresponding to time offsets	radians	-2*pi	2*pi
sunZenithLimitTag	tag specifying basis of day/night determination Allowed values: 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.	N/A	N/A	N/A

OUTPUTS:

Table 6-238. PGS_CSC_DayNight Outputs

Name	Description	Units	Min	Max
afterDark	array of answers: Array values will be either PGS_TRUE or PGS_FALSE, according to the tag definition. PGS_TRUE means point is in night, PGS_FALSE means point is in daylight or twilight.	Boolean	see DESCRIPTION	see DESCRIPTION

RETURNS:

Table 6-239. PGS_CSC_DayNight Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_NO_LEAP_SECS	No leap second value available in table for at least one of the input offset times; a linear approximation was used to get value
PGSCSC_E_INVALID_LIMITTAG	Invalid sunZenithLimitTag
PGSCSC_E_BAD_ARRAY_SIZE	numValues (and array size) is less than zero
PGSCSC_W_ERROR_IN_DAYNIGHT	An error occurred in computing at least one afterDark value
PGSCSC_W_BAD_TRANSFORM_VALUE	Invalid ECItoECR transformation
PGSCSC_W_BELOW_HORIZON	Sun is below horizon
PGSCSC_W_PREDICTED_UT1	At least one of the values obtained from the utcpole.dat file is 'predicted'
PGSTD_E_NO_UT1_VALUE	No UT1–UTC correction available
PGSTD_E_BAD_INITIAL_TIME	Initial input time cannot be deciphered
PGSCBP_E_TIME_OUT_OF_RANGE	Start UTC time is not in the range of the planetary ephemeris file (de200.eos)
PGSCBP_E_UNABLE_TO_OPEN_FILE	Ephemeris file cannot be opened
PGSMEM_E_NO_MEMORY	No memory available to allocate vectors
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
PGSt_integer      counter;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                  {3600.0,7200.0,10800.0};
PGSt_double       latitude[ARRAY_SIZE]= {0.5,0.75,0.90};
PGSt_double       longitude[ARRAY_SIZE] = {1.0,2.0,3.0};
PGSt_tag          sunZenithLimitTag = PGSd_CivilTwilight;
PGSt_boolean      afterDark[ARRAY_SIZE];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CSC_DayNight(numValues,asciiUTC,offsets,
```

```
                                    latitude,longitude,
                                    sunZenithLimitTag,
                                    afterDark);

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter <= numValues)
{
    printf("Offset: %lf   Latitude:%lf   Longitude:%lf
      Day/Night:%u", offset[counter],
            latitude[counter], longitude[counter],
            afterDark[counter]);
            counter++;
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_daynight
parameter         (array_size=3)
integer           returnstatus
integer           counter
integer           numvalues
character*27      asciiutc
double precision  offsets(array_size)
double precision  latitude(array_size)
double precision  longitude(array_size)
integer           sunzenithlimittag
integer           afterdark(array_size)

data offsets/3600.0,7200.0,10800.0/
data latitude/0.5,0.75,0.90/
data longitude/1.0,2.0,3.0/
numvalues = array_size
asciiutc = '1991-01-01T11:29:30'
sunzenithlimittag = pgsd_civiltwilight

returnstatus = pgs_csc_daynight(numvalues,asciiutc,offsets,
                                    latitude,longitude,
                                    sunzenithlimittag,
                                    afterdark)
```

```
      if(returnstatus .ne. pgs_s_success) go to 90

      write(6,*) asciiutc
      if(numvalues.eq.0) numvalues = 1
      do 40 counter = 1,numvalues,1
         write(6,*)offsets(counter),latitude(counter),
                   longitude(counter),afterdark(counter)

   40 continue

   90 write(6,99)returnstatus

   99 format('ERROR:',I50)
```

NOTES: If there is an error in computing one or more of the afterDark values, which does not affect the computation of the other values for the input offset times, it is set to the returnStatus value.

An Earth model tag is not needed because the latitude is geodetic. Input latitude values should be based on an Earth model (flattening) consistent with that used for other data analysis and processing for the same spacecraft.

User supplies one of the four sunZenithLimitTags as part of the input information. Users wishing to know if a point is in Nautical Twilight or darker should use the Civil Night tag; those wishing to determine whether the point is after the start of Astronomical Twilight should use the Nautical Night tag. An example of tag usage is the following: if the tag is set to Nautical night and the Sun zenith angle is less than 102 degrees, afterDark will be false; if 102 degrees or more it will be true.

TIME ACRONYMS:

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REQUIREMENTS: PGSTK–0860, PGSTK–0930

Calculate Nutation Angles

NAME: **PGS_CSC_wahr2( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGS_CSC_wahr2(
        PGSt_double ddjd[2],
        PGSt_double dvnut[4])
```

FORTRAN:

```
include 'PGS_SMF.f'

integer function pgs_csc_wahr2(ddjd,dvnut)
        double precision    ddjd(2)
        double precision    dvnut(4)
```

DESCRIPTION: Calculates nutation angles delta psi and delta epsilon, and their rates of change, referred to the ecliptic of date, from the Wahr series.

INPUTS:

Table 6-240. PGS_CSC_wahr2 Inputs

Name	Description	Units	Min	Max
ddjd[2]	Barycentric Dynamical Time as a Julian Date	N/A	ANY	ANY
ddjd[0]	half-integral Julian day			
ddjd[1]	Julian day fraction			

OUTPUTS:

Table 6-241. PGS_CSC_wahr2 Outputs

Name	Description	Units	Min	Max
dvnut[0]	nutation in longitude	radians	-0.01	0.01
dvnut[1]	nutation in obliquity	radians	-0.001	0.001
dvnut[2]	nutation rate in longitude	radians/sec	-1.16e-1	+1.16e-11
dvnut[3]	nutation rate in obliquity	radians/sec	-1.16e-13	+1.16e-13

RETURNS:

Table 6-242. PGS_CSC_wahr2 Returns

Return	Description
PGS_S_SUCCESS	Successful return

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       dvnut[4];

returnStatus = PGS_CSC_wahr2(jedTDB,dvnut);

** do something with shiny new nutation angles and rates **
                  :
                  :
```

FORTRAN:

```
implicit none

integer           pgs_csc_wahr2
integer           returnstatus
double precision  jedtdb(2)
double precision  dvnut(4)

data jedtdb/2449720.5,0.25/

returnstatus = pgs_csc_wahr2(jedtdb,dvnut)

!            do something with shiny new nutation angles and rates
                        :
                        :
```

NOTES:

From table 1, "proposal to the International Astronomical Union (IAU) working group on nutation," John M. Wahr and Martin L. Smith (1979) subroutine to compute nutation angles and rates from expressions given in Supplement to Astronomical Almanac 1984, S21–S26. Ref: P.K. Seidelmann, V.K. Abalakin, H. Kinoshita, J. Kovalevsky, C.A. Murray, M.L. Smith, R.O. Vicente, J.G. Williams, Ya. S. Yatskiv: 1982, "1980 IAU Theory of Nutation", Celestial Mechanics Journal, vol 27., p. 79–105

Changes to code prior to acquisition for ECS project:

Lieske 3/91. NUTATION in the IAU J2000 system. Univac version obtained from Myles Standish, (subroutine WAHR) who had obtained it from USNO. Re-ordered terms to match Astronomical Almanac 1984 table S23-S25 and corrected the rate for dPsi in the 0 0 2 -2 2 term. Eliminated the equivalencies, common block and added necessary SAVEs. Corrected the fundamental angles (L, L', F, D, Node) to match Almanac.

Acquired from E. Myles Standish, JPL, 12/93 by Peter Noerdlinger. This is not JPL certified code. Please do not modify the names of the variables in this code. It is heritage code and we may receive updates. We may also receive other related code with the same names for variables.

Users concerned with speed may wish to avoid repeated calls where possible. In this regard, the rates that are provided by Wahr2 can be used either for estimating the error of using nearby times, or for short term

extrapolation. Note that in the original JPL code the rates issued by wahr2 are in radians per day; this routine returns the rates as radians per second.

REQUIREMENTS: PGSTK–0916, PGSTK–0930, PGSTK–1050

Get Greenwich Hour Angles

NAME: **PGS_CSC_GreenwichHour( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GreenwichHour(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         hourAngleGreenw[])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_greenwichhour(numvalues, asciiutc, offsets(*),
                      houranglegreenw(*))
        integer         numvalues,
        character*27        asciiutc,
        double precision    offsets(*),
        double precision    houranglegreenw(*)
```

DESCRIPTION: This function computes hour angle of the Vernal Equinox at the Greenwich meridian, accepting an input start time plus an array of time offsets.

INPUTS:

Table 6-243. PGS_CSC_GreenwichHour Inputs

Name	Description	Units	Min	Max
numValues	number of input time offsets	N/A	0	any
asciiUTC	UTC start time in CCSDS ASCII Time Code A or B format	N/A	see Notes	see Notes
offsets	array of time offsets	seconds	Max and Min such that asciiUTC + offset is between asciiUTC Min and Max values	

OUTPUTS:

Table 6-244. PGS_CSC_GreenwichHour Outputs

Name	Description	Units	Min	Max
hourAngleGreenw	array of values of the hour angle of the Vernal Equinox at Greenwich; a value of 999999.0 is returned for invalid offset times	hours	0	24

RETURNS:

Table 6-245. PGS_CSC_GreenwichHour Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_ERRORS_IN_GHA	An error occurred in computing at least one Greenwich hour angle
PGSCSC_W_PREDICTED_UT1	Data in utcpole.dat file is predicted (not final) value for at least one input time
PGSTD_E_TIME_VALUE_ERROR	Error in input time value
PGSTD_E_TIME_FMT_ERROR	Error in input time format
PGSTD_E_NO_LEAP_SECS	No leap seconds correction is available in leapsec.dat file for at least one of the input times/offsets
PGS_E_TOOLKIT	Something unexpected happened, execution of function ended prematurely

EXAMPLES:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
PGSt_integer      counter;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE]=
                  {3600.0,7200.0,10800.0};
PGSt_double       hourAngleGreenw[ARRAY_SIZE];

numValues=ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CSC_GreenwichHour(numValues,asciiUTC,
                                     offsets,
                                     hourAngleGreenw);
```

```
if(returnStatus != PGS_S_SUCCESS)
{
** test errors,
   take appropriate
   action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter < numValues)
{
    printf("Offset: %lf   Hour Angle:%lf",offset[counter],
        hourAngleGreenw[counter]);
    counter++;
}
```

FORTRAN:

```
implicit none

parameter (array_size=3)
integer           pgs_csc_greenwichhour
integer           array_size
integer           returnstatus
integer           counter
integer           numvalues
character*27      asciiutc
double precision  offsets(array_size)
double precision  houranglegreenw(array_size)

data offsets/3600.0,7200.0,10800.0/
array_size = 3
numvalues = array_size
asciiutc = '1991-01-01T11:29:30'

returnstatus = pgs_csc_greenwichhour(numvalues,asciiutc,
                                     offsets,
                                     houranglegreenw)

if(returnstatus .ne. pgs_s_success) go to 90
write(6,*) asciiutc
if(numvalues.eq.0) numvalues = 1
do 40 counter = 1, numvalues,1
write(6,*)offsets(counter),houranglegreenw(counter)

40 continue

90 write(6,99)returnstatus

99 format('ERROR:',A50)
```

NOTES: Historically, UT1 was used as a measure of time, but since 1958 it has served only as a measure of Earth rotation. The only real difference between UT1 and Greenwich Mean Sidereal Time (GMST) is that UT1 measures Earth rotation in regards to the vector from Earth center to the mean sun (a fictitious point that traverses the celestial equator at the same mean rate that the sun apparently traverses the ecliptic), while GMST measures Earth rotation relative to the vernal equinox. Essentially, the value of GMST in radians is larger than that of UT1 in radians by the ratio of the mean solar day to the sidereal day; however, there are small correction terms due to precession. The equation used in function PGS_TD_gmst( ) is valid for the period 1950 to well past 2000, as long as the definition of UT1 and the reference equinox (J2000) are not changed. The basic limitation is the accuracy of UT1. Users obtaining UT1 from the SDP Toolkit should observe time limitations in the function PGS_TD_UTCtoUT1( ).

TIME ACRONYMS:

GMST is: Greenwich Mean Sidereal Time
TAI is: International Atomic Time
UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

REFERENCE FOR TIME:

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

REQUIREMENTS: PGSTK–0770

Get Zenith and Azimuth of an ECR Vector at the Look Point

NAME: **PGS_CSC_ZenithAzimuth( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ZenithAzimuth(
        PGSt_double     vectorECR[3],
        PGSt_double     latitude,
        PGSt_double     longitude,
        PGSt_double     altitude,
        PGSt_tag        vectorTag,
        PGSt_boolean    zenithOnlyFlag,
        PGSt_boolean    refractFlag,
        PGSt_double     *zenith,
        PGSt_double     *azimuth
        PGSt_double     *refraction)
```

FORTRAN:

```
include'PGS_CSC.f'
include'PGS_CSC_4.f'
include'PGS_SMF.f'

integer function
pgs_csc_zenithazimuth(vectorecr,latitude,longitude,altitude,
                                        vectortag,zenithonlyflag,
                                        refractflag,zenith,azimuth,
                                        refraction)
        double precision    vectorECR[3]
        double precision    latitude
        double precision    longitude
        double precision    altitude
        integer             vectortag
        integer             zenithonlyflag
        integer             refractflag
        double precision    zenith
        double precision    azimuth
        double precision    refraction
```

DESCRIPTION: Computes the zenith and the azimuth of a vector at the look point. This tool allows for refraction if desired.

INPUTS:

Table 6-246. PGS_CSC_ZenithAzimuth Inputs

Name	Description	Units	Min	Max
vectorECR	ECR vector whose zenith & azimuth is desired (in case of PGSD_MOON do not use a unit vector!—see NOTES)	meters or unit vector	N/A	N/A
latitude	geodetic latitude	radian	-pi/2	pi/2
longitude	longitude	radian	-2*pi	2*pi
altitude	altitude off the geoid (altitude is an input only when refracFlag is PGS_TRUE see NOTES).	meters	-2000	80000
vectorTag	PGSd_CB, PGSd_moon, or PGSd_Look or a CB identifier (see NOTES)	N/A	N/A	N/A
zenithOnlyFlag	omit azimuth calculation	N/A	N/A	N/A
refracFlag	turns on refraction	N/A	N/A	N/A

OUTPUTS:

Table 6-247. PGS_CSC_ZenithAzimuth Outputs

Name	Description	Units	Min	Max
zenith	zenith angle	radian	0	1.6755 (96 deg)
azimuth	azimuth E from N	radian	-pi	+pi
refraction	increase of zenith angle due to refraction	radian	0	0.1

RETURNS:

Table 6-248. PGS_CSC_ZenithAzimuth Returns

Return	Description
PGS_S_SUCCESS	Successful execution
PGSCSC_W_BELOW_HORIZON	Warning indicating the object is below horizon
PGSCSC_W_UNDEFINED_AZIMUTH	The object is at the zenith. In this case azimuth is not calculated
PGSCSC_W_NO_REFRACTION	No refraction calculation done due to errors
PGSCSC_E_INVALID_VECTAG	The input vector tag is not PGSd_CB, PGSd_MOON, PGSd_LOOK or a celestial body identifier
PGSCSC_E_LOOK_PT_ALTIT_RANGE	Look point altitude not reasonable
PGSCSC_E_ZERO_INPUT_VECTOR	The input vector has zero length
PGS_E_TOOLKIT	Unknown error occurred

EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       vectorECR[3];
PGSt_double       latitude  = 0.2;
PGSt_double       longitude = 0.1;
PGSt_double       altitude = 0.0;
PGSt_tag          vectorTag = PGSd_LOOK;
PGSt_boolean      zenithOnlyFlag = PGS_FALSE;
PGSt_boolean      refractFlag = PGS_FALSE;
PGSt_double       zenith;
PGSt_double       azimuth;
PGSt_double       refraction;

vectorECR[2] = -0.26;
vectorECR[1] = 0.0;
vectorECR[0] = sqrt(1 - vectorECR[2]*vectorECR[2]);

returnStatus = PGS_CSC_ZenithAzimuth(vectorECR,latitude,
                                     longitude,altitude,
                                     vectorTag,
                                     zenithOnlyFlag,
                                     refractFlag,&zenith,
                                     &azimuth,&refraction)

 do some error handling
 if desired, convert zenith and azimuth to degrees
printf("zenith angle = %lf,  azimuth = %lf\n", zenith,
        azimuth);
```

FORTRAN:

```
implicit none

integer           pgs_csc_zenithazimuth
double precision  look[3]
double precision  latitude
double precision  longitude
double precision  altitude
integer           zenithonlyflag
integer           refractflag
double precision  zenith
double precision  azimuth
double precision  refraction
integer           vectortag,returnstatus

vectortag = pgsd_look
latitude  = 0.2D0
longitude = -0.3D0
```

```
altitude  = 0.0D0
zenithonlyflag = PGS_FALSE
refractflag    = PGS_FALSE

look[3] = -0.26;
look[2] = 0.0;
look[1] = sqrt(1 - look[3]*look[3]);

returnstatus = pgs_csc_zenithazimuth(look,latitude,
                                     longitude,altitude,
                                     vectortag,
                                     zenithonlyflag,
                                     refractflag,zenith,
                                     azimuth,refraction)

C       do some error handling

C       if desired, convert zenith and azimuth to degrees
```

NOTES: The vectorECR vector must be in ECR coordinates. For celestial bodies, it is the vector from Earth to the celestial body. It can be obtained by getting the ECI vector to the body from PGS_CBP_Earth_CB_Vector(), and transforming that vector to ECR rectangular coordinates with PGS_CSC_ECItoECR().

The "look vector" (which could as well be called the "boresight vector") is an ECR vector from the instrument to the point being viewed)"look point."). To obtain the zenith and azimuth of the look vector, the vectorTag must be set to PGSd_LOOK (this allows for the reversed sense of such a vector, which represents a line of sight above the horizon when pointing down). If desired, the unit look vector can be obtained and saved from PGS_CSC_GetFOV_Pixel( ); this will achieve very good performance, as the ECR look vector is calculated there. If the ECR look vector has to be constructed (for example, when starting with already-geolocated data), this can be done, for example, as follows: Convert the latitude, longitude and altitude of the look point to ECR rectangular coordinates with PGS_CSC_GEOtoECR(). Then obtain the ECI spacecraft position from PGS_CSC_Ephem_Attit(), and convert it to ECR with PGS_CSC_ECItoECR(). Finally, subtract the last result from the first. The geometry is illustrated in Fig 6-4:

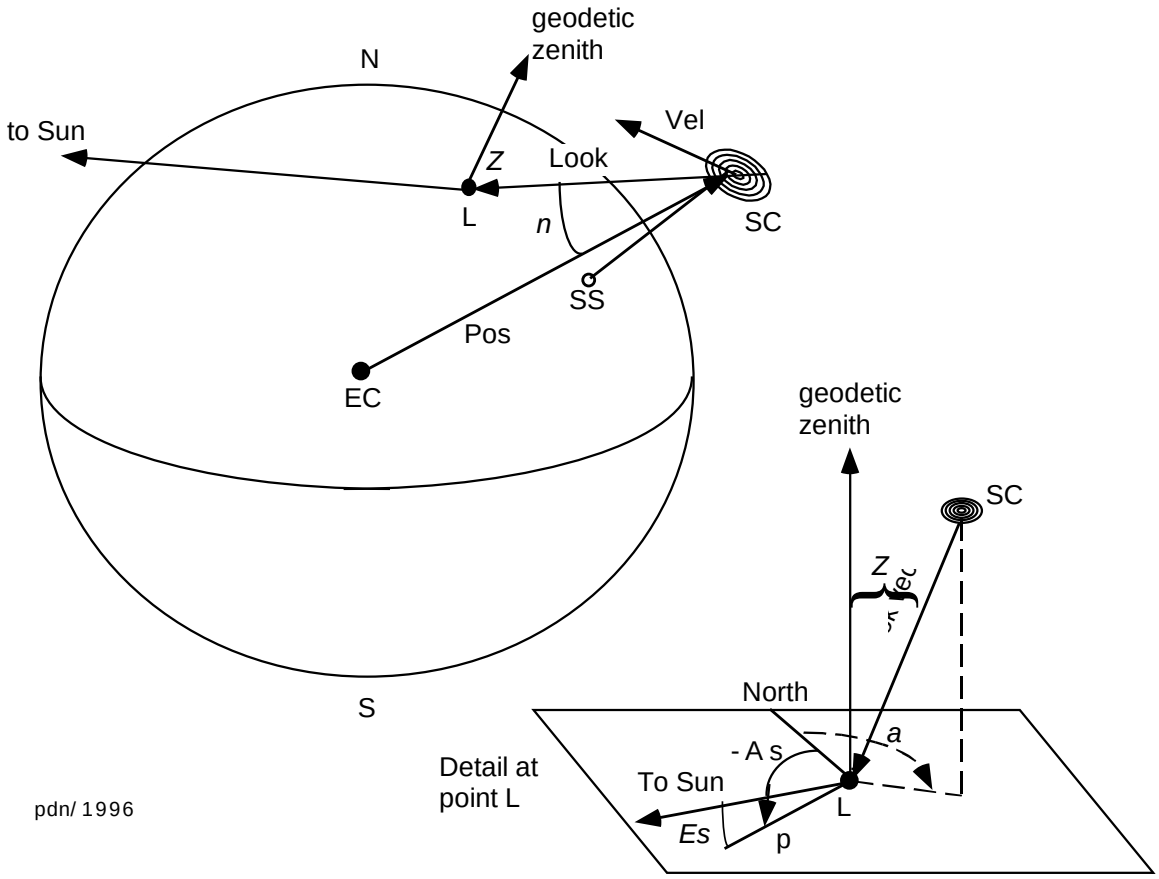


Figure 6-4. Geometry of the Viewing and Sun Vectors

Notation:
SC = spacecraft
EC = Earth Center
Pos = position vector
Vel = velocity vector
SS = subsatellite point
L = Lookpoint
n = nadir angle
Es = elevation of the Sun = (π/2 - solar zenith angle).
p = projection of Sun vector on horizontal
As = solar azimuth. All azimuths are measured East from North
Z = zenith angle of look vector. *a* = azimuth of the Look vector.

Note that because of Earth curvature, generally *Z* > *n*. Also, in this diagram m,*n* is referenced to geocentric, SS to geodetic, a small difference. In other contexts, however, "nadir angle" could refer to the

angle between a direction and either geocentric, geodetic, or nominal spacecraft nadir.

Note: In ECR coordinates, the Look Vector is also called the ECR Pixel Vector, but in SC coordinates, it is called the SC Pixel Vector.

If the zenith and azimuth of a distant celestial body (such as the sun or a planet) are desired, the user may supply PGSd_CB or any of the identifiers: PGSd_SUN, PGSd_MERCURY, PGSd_VENUS, PGSd_MARS, PGSd_JUPITER, PGSd_SATURN, PGSd_URANUS, PGSd_NEPTUNE, or PGSd_PLUTO. This is purely a convenience for users doing other calculations with a CB identifier; the action of the function is in all cases the same—it finds the zenith and azimuth of the vector at the look point, without regard to parallax (i.e., the vector from Earth center to the Celestial body is regarded as unchanged due to the displacement of the look point from Earth center).

In the case of the PGSd_MOON, the geocentric parallax is appreciable, meaning that its apparent position is, in general, different as viewed from Earth center or from the look point. The difference can be as large as a degree. Therefore, in this case, a parallax correction is made. It is essential, in this case, of course, that the PGSd_MOON vector be supplied in meters. In this case, the input vector should be the Earth to PGSd_MOON vector defined from Earth center (geocentric), as obtained, for example, from the PGS_CBP_Earth_CB_Vector( ) tool.

In all other cases, the input vector can be in any units, including normalized (unit vector).

Users wishing to take into account the minuscule parallax correction for the sun, or the correction for some other chosen body such as an asteroid, could simply label the vector as PGSd_MOON. (For the sun, the correction is only ~ 2.5 millidegrees.)

Refraction by the atmosphere is calculated if the flag is set to PGS_TRUE. This calculation approximately corrects, in the visual band, for the fact that any line of sight, such as the sun, moon, or look vector is bent by the atmosphere.

If the vector is well below the horizon, a warning is returned and no azimuth calculation is done. The present algorithm is fairly forgiving for points slightly below the horizon (to 96 degrees), in order that the user interested in the location of the glow before sunrise or after sunset can find its azimuth; it is user responsibility to take special action between 90 degrees and 96 degrees if these data are not wanted.

The altitude is required only if refraction is to be calculated, and its only effect is to change the mean density of the atmosphere in the refraction function.

If the zenith only flag is defined by the user to be PGS_TRUE the function will run faster but will not calculate the azimuth.

If the azimuth is requested but the zenith angle is < 0.026 deg, it is deemed that the azimuth calculation is unreliable, because variations in the local vertical as determined from the geoid, and variable refraction in the atmosphere dominate at that level. The azimuth is returned as 0.0 and the warning PGSCSC_W_UNDEFINED_AZIMUTH is returned

The calculation herein is entirely independent of the Earth model except for the parallax correction, where WGS84 is assumed, and any difference in other models introduces negligible error. The use of geodetic latitude as input guarantees that the rest of the algorithm is independent of Earth model.

REQUIREMENTS: PGSTK–1091

Find Point of Closest Miss and Surface Point

NAME: **PGS_CSC_GrazingRay( )**

SYNOPSIS:

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GrazingRay(
        char            earthEllipsTag[50],
        PGSt_double     posECR[3],
        PGSt_double     ray[3],
        PGSt_double     *latitude,
        PGSt_double     *longitude,
        PGSt_double     *missAltitude,
        PGSt_double     *slantRange,
        PGSt_double     posNEAR[3],
        PGSt_double     posSURF[3])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'

integer function pgs_csc_grazingray(
        earthellipstag,pos,ray,latitude,longitude,
        missaltitude,slantrange,posnear,possurf)

character*49        earthellipstag
double precision    posecr(3)
double precision    ray(3)
double precision    latitude
double precision    longitude
double precision    missaltitude
double precision    slantrange
double precision    posnear(3)
double precision    possurf(3)
```

DESCRIPTION: For rays that miss Earth limb, this function finds the nearest miss point on the ray and the corresponding surface point. For rays that strike the Earth, it finds instead the midpoint of the chord of the ray within the ellipsoid and the surface point of intersection nearest the observer.

INPUTS:

Table 6-249. PGS_CSC_GrazingRay Inputs

Name	Description	Units	Min	Max
earthEllipsTag	tag selecting Earth ellipsoid model (default is WGS84)	N/A	N/A	N/A
posECR	ECR Spacecraft Position	meters	N/A	see NOTES
ray[3]	unit vector along the line of sight, in ECR coordinates	N/A	-1 per component	+1 per component

OUTPUTS:

Table 6-250. PGS_CSC_GrazingRay Outputs

Name	Description	Units	Min	Max
latitude	geodetic latitude of posNEAR (q.v. below)	radians	-pi/2	pi/2
longitude	longitude of posNEAR (q.v. below)	radians	-pi	pi
missAltitude	altitude of posNEAR (q.v. below)	meters	N/A	N/A
slantRange	range to posNEAR (q.v. below)	m/s	-7000	7000
posNEAR[0]	X coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES)	meters	N/A	N/A
posNEAR[1]	Y coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES)	meters	N/A	N/A
posNEAR[2]	Z coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES)	meters	N/A	N/A
posSURF[0]	X coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid, (in the case that it does not miss)	meters	N/A, but normally < 6378140	N/A, but normally > -6378140
posSURF[1]	Y coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid (in the case that it does not miss)	meters	N/A, but normally < 6378140	N/A, but normally > -6378140
posSURF[2]	Z coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid (in the case that it does not miss)	meters	N/A, but normally < 6378140	N/A, but normally > -6378140

RETURNS:

Table 6-251. PGS_CSC_GrazingRay Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCSC_W_SUBTERRANEAN	User provided a subterranean position for the spacecraft
PGSCSC_W_HIT_EARTH	Line of Sight struck the Ellipsoid
PGSCSC_W_LOOK_AWAY	Line of sight points away from Earth
PGSCSC_W_ERROR_IN_GRAZINGRAY	Generic return for warning in lower level function
PGSCSC_W_SPHERE_BODY	Using a spherical Earth model
PGSCSC_W_LARGE_FLATTENING	Issued if flattening factor is greater than 0.01
PGSCSC_W_DEFAULT_EARTH_MODEL	Default Earth model was used - user's model not found
PGSCSC_W_ZERO_PIXEL_VECTOR	Zero length ray vector supplied (terminates execution)
PGSCSC_E_BAD_EARTH_MODEL	Equatorial or polar radius less than or equal to 0.0 or the model defines a prolate Earth
PGS_E_TOOLKIT	Something unexpected happened—execution aborted

EXAMPLES:[1]

```
C:          #include <PGS_CSC.h>
            PGSt_SMF_status   returnStatus
                  char              earthEllipsTag[50];
                  PGSt_double       posECR[3];
                  PGSt_double       ray[3];
                  PGSt_double       latitude;
                  PGSt_double       longitude;
                  PGSt_double       missAltitude;
                  PGSt_double       slantRange;
                  PGSt_double       posNEAR[3];
                  PGSt_double       posSURF[3];

            strcpy(earthEllipsTag,"GEM-10B");
            posECR[0] = 4077000.0;
                  posECR[1] = 5000000.0;
                  posECR[2] = -3200000.0;
            ray[0] = 0.0002;
                  ray[1] = -1.0;
                  ray[2] = -0.422;
            returnStatus = PGS_CSC_GrazingRay(
                  earthEllipsTag,posECR,ray,&latitude,&longitude,
                  &missAltitude,&slantRange, posNEAR,
                  posSURF);
```

[1] Note: As is Toolkit standard, to avoid possible interface problems to outside software that may support different word lengths, the Toolkit renormalizes all unit input vectors at the same time it checks for zero length input vectors. Therefore the inputs shown here are unnormalized.

```
            printf("Longitude %f\n",longitude);
                  printf("Latitude: %f\n",latitude);
                  printf("Altitude: %f\n",missAltitude);
                  printf("Slant Range: %f\n",slantRange);
            if(returnStatus == PGS_S_SUCCESS)
                  {
                    printf("Point on Ray Nearest Earth: %f, %f, %f\n",
                    posNEAR[0],posNEAR[1],posNEAR[2]);
                    printf("Point on Surface Nearest Ray: %f, %f, %f\n",
                    posSURF[0],posSURF[1],posSURF[2]);
                  }
            else if(returnStatus == PGSCSC_W_HIT_EARTH)
                  {
                  printf("Midpoint of Ray Chord in Earth: %f, %f, %f\n",
                     posNEAR[0],posNEAR[1],posNEAR[2]);
                  printf("Line of Sight Strikes Earth at: %f, %f, %f\n",
                              posSURF[0],posSURF[1],posSURF[2]);
                  }
            else
                  {
              ** test errors,
                  take appropriate
                   action **
                  }

FORTRAN:
            include 'PGS_SMF.f'
            include 'PGS_CSC_4.f'
            implicit none
            integer           pgs_csc_grazingray
            integer           returnstatus
            character*19      earthellipstag
            double precision  posecr(3)
            double precision  ray(3)
            double precision  latitude
            double precision  longitude
            double precision  missaltitude
            double precision  slantrange
            double precision  posnear(3)
            double precision  possurf(3)

            posecr(1) = 4077000.0
            posecr(2) = 5000000.0
            posecr(3) = -3200000.0
            ray(1) = -0.0002
            ray(2) = -1.0
            ray(3) = -0.422
            returnstatus = pgs_csc_grazingray(
            earthellipstag,posecr,ray,latitude,longitude,
                  missaltitude,slantrange, posnear,
                  possurf)
            print*,'Longitude: ',longitude
            print*,'Latitude: ',latitude
            print*,'Slant Range: ',slantrange
            print*,'Altitude: ',missaltitude

      if(returnStatus .eq. PGS_S_SUCCESS) then
C ampersands & below are continuation marks in column
```

```
         print*,'Point on Ray Nearest Earth: X = ',
         posnear(1),'  Y = ',posnear(2),' Z = ', posnear(3)
         print*,'Point on Surface Nearest Ray: X = ',
     &   possurf(1),'  Y = ',possurf(2),' Z = ', possurf(3)
       else if (returnStatus .eq. PGSCSC_W_HIT_EARTH) then
         print*,'Midpoint of Ray Chord in Earth: X = ',
     &posnear(1),'  Y = ',posnear(2),' Z = ', posnear(3)
         print*,'Line of Sight Strikes Earth at: X = ',
     &possurf(1),'  Y = ',possurf(2),' Z = ', possurf(3)
       else
C  ** test errors, take appropriate action **
       endif
       print*,err,msg
```

NOTES: For a line of sight ("ray") that misses Earth limb, this tool calculates the rectangular coordinates of the point Q of closest approach to the Earth and the slant range to Q. It also obtains the latitude and longitude of the surface point P nearest Q (and therefore nearest to the ray) and the geodetic altitude of Q above P (Q and P have the same longitude and geodetic altitude). When the ray, instead, intersects the Earth ellipsoid, the rectangular coordinates of Q are replaced by those of a point halfway between the two "pierce points" where the ray intersects the ellipsoid. The intent is to provide a point with the nearly the same latitude and longitude as the point closest to the Earth's surface on a ray from a background object (such as the Sun) that is actually refracted round the Earth. When the ray intersects the Earth, the latitude and longitude of P are also replaced by those of the nearest pierce point, i.e. where the instrument is looking, and the slant range is replaced by the range to that point. Furthermore, the return value PGSCSC_W_HIT_EARTH issues. If the ray, instead, points *away* from the Earth ellipsoid, the altitude output variable is set to PGSd_GEO_ERROR_VALUE and the return value to PGSCSC_W_LOOK_AWAY, in either case - ray missing the ellipsoid or ray striking it. The return PGSCSC_W_ZERO_PIXEL_VECTOR terminates execution, even though it is a "warning" level only; the "warning" status was defined for this message to support certain tools that process arrays of pixels at once, in order that if a few pixel vectors were bad, the remainder could be processed. The same return is reused here, but promoted to have a fatal result.

If an invalid earthEllipsTag is input, the program will use the WGS84 Earth model by default.

REQUIREMENTS: PGSTK-1085

6.3.4.11 CSC Functions

PGS_CSC_DayNight

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ECItoECR

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ECItoORB

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ECItoSC

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ECRtoECI

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ECRtoGEO

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_EarthOccult

Test for earth occultation of a celestial body in the field of view. The test is in three phases. The first phase does not depend on the CB at all - it is just a check if the Earth fills the field of view. The second test (exercised only if the first fails to find total occultation) determines if the celestial body is behind the Earth. If the second test fails, the vector in SC coordinates that points at the part of the CB most distant from the Earth center is returned so that the calling function can determine if the Earth's bulge (difference in radius over that of an inscribed sphere) occults the CB.

PGS_CSC_Earthpt_FixedFOV

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_Earthpt_FOV

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_EulerToQuat

Transforms Euler angles to Quaternions.

PGS_CSC_FOVconicalHull

A circular cone is drawn around the FOV and a check is made as to whether the candidate point is inside it before going any further. The function has two purposes:

a. it will speed up tasks by obviating complicated algorithms for points well away from the FOV

b. it will enable detection and rejection of FOV specifications outside our present algorithmic limits. [Present software does not reliably handle fields of view more than 180 degrees across.]

PGS_CSC_GEOtoECR

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_GetEarthFigure

This tool gets the equatorial and polar radii from the earthfigure.dat file for the earth model input.

PGS_CSC_GetFOV_Pixel

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_GreenwichHour

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_J2000toTOD

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_LookPoint

Solves the look point equation. method:

Solve the quadratic equation

x = p + d*u

where x must lie on an ellipsoid, for the slant range, d, corresponding to the intersection of the extended look vector, u, with the surface of the earth. Then compute x directly.

PGS_CSC_Norm

This tool computes the norm of a 3-vector.

PGS_CSC_ORBtoECI

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_ORBtoSC

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_PointInFOVgeom

For each input point, the function does the processing to determine if a point is in the field of view and returns a flag indicating whether the point is in the field of view.

PGS_CSC_QuatToEuler

This function gets Euler angles from a quaternion.

PGS_CSC_SCtoECI

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_SCtoORB

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_SpaceRefract

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_SubSatPoint

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_SubSatPointVel

This tool finds the North and East components of the velocity of the subsatellite point and the rate of change of spacecraft altitude.

PGS_CSC_TODtoJ2000

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_TiltYaw

Obtains the tipped orbital (geodetic nadir) to orbital transformation quaternion.

PGS_CSC_UT1_update

updates the file “utcpole.dat”

PGS_CSC_UTC_UT1Pole

This tool accesses the file 'utcpole.dat' and extracts using interpolation the x,y pole postion in seconds of arc and the difference of UT1 and UTC, given an input Julian date.

PGS_CSC_ZenithAzimuth

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

PGS_CSC_crossProduct

Cross product of vectors

PGS_CSC_dotProduct

Dot product of vectors

PGS_CSC_getECItoORBquat

This function returns a quaternion describing the rotation from the Earth Centered Inertial (ECI) reference frame to the Orbital (ORB) reference frame. That is the quaternion returned will transform a vector in the ECI frame to the equivalent vector in the ORB frame.

PGS_CSC_getORBtoECIquat

This function returns a quaternion describing the rotation from the Orbital (ORB) reference frame to the Earth Centered Inertial (ECI) reference frame. That is the quaternion returned will transform a vector in the ORB frame to the equivalent vector in the ECI frame.

PGS_CSC_getQuats

Converts a transformation matrix to a quaternion

PGS_CSC_nutate2000

This tool transforms a vector under nutation between Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB) and True Celestial Coordinates of date.

PGS_CSC_precs2000

This tool precesses a vector from Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB).

PGS_CSC_quatMultiply

This file contains the function PGS_CSC_quatMultiply(). This function multiplies two quaternions, using a short algorithm with 11 multiplications and 19 additions.

PGS_CSC_quatRotate

This function transforms a vector from one coordinate system to a rotated coordinate system with a common origin, where the rotation is defined by a quaternion.

PGS_CSC_quickWahr

Wahr nutation with extrapolation for up to 1/2 hour (valid to microseconds of arc)

PGS_CSC_wahr2

Calculates nutation angles delta psi and delta epsilon, and their rates of change, referred to the ecliptic of date, from the Wahr series.

6.3.5 Geo–Coordinate Transformation Tools

The geo–coordinate transformation tools are required to support the bi–directional transformation between geographic coordinates and various standard map projection frames. The tools are designed to give rapid coordinate transformations for many points. The tool provides transformations from geodetic latitude and longitude as output from the geolocation tool into the required map projection frames.

There are initialization routines for each set of transformations (PGS_GCT_Init). PGS_GCT_Init routine is used to “save” in memory the projection parameters and to pre–calculate any variables that are required in all subsequent transformations. Following the initialization of a projection

there are routines for the forward and reverse transformations combined into a single interface (PGS_GCT_Proj).

The tool may be used to perform many transformations in different projections within the same executable program. If the same projection is required twice to perform transformations with two different sets of parameters, then the initialization routine has to be called before each set of transformations.

Every effort should be made to standardize projection definitions throughout the project to enable data sets to be generated in a consistent manner.

In addition to the standard transformations of latitude and longitude to map projection specific transformations from one projection to another are required, these are treated in a similar manner as discussed below.

The tool is based on the commonly available packages general cartographic transformation package (GCTP) for coordinate transformation; this package is based on the projections described by Snyder. *Map Projections—A Working Manual*—J.P. Snyder, USGS professional paper 1395, 1987.

Initialize Given Projection Parameters

NAME: **PGS_GCT_Init( )**

SYNOPSIS:

C:

```
#include <PGS_GCT.h>

PGSt_SMF_status
PGS_GCT_Init(
        PGSt_integer        projId,
        PGSt_double         projParam[],
        PGSt_integer        directFlag)
```

FORTRAN:

```
include"PGS_GCT.f"
include"PGS_GCT_12.f"
include"PGS_SMF.f"

integer function pgs_gct_init( projid, projparam, directflag)
        integer             projid
        double precision(30)  projparam
        integer             directflag
```

DESCRIPTION: This tool provides a general interface to perform geo–coordinate transformations in the forward/inverse directions. In general the tool requires a projection id, location of input data vectors and the direction of the conversion. PGSd_UTM projection is a special case for which zone value is also needed to define a point.

INPUTS:

Table 6-252. PGS_GCT_Init Inputs

Name	Description	Units	Min	Max
projId	projection code	none	1	#defined
projParam	projection parms	rad, m if latitude if longitude	-90(PI/180) -PI	90(PI/180) PI
directFlag	forward/inverse	none	PGSd_GCT_FORWARD	PGSd_GCT_INVERSE

OUTPUTS: None

RETURNS:

Table 6-253. PGS_GCT_Init Returns

Return	Description
PGS_S_SUCCESS	
PGSGCT_E_NO_DATA_FILES	Data files for state plane could not be found
PGSGCT_E_GCTP_ERROR	Error has occurred in the GCTP lib
PGSGCT_E_BAD_INC_ANGLE	Invalid inclination angle in the Space Oblique Mercator (SOM) projection
PGSGCT_E_BAD_RADIUS	Invalid radius
PGSGCT_E_BAD_MINOR_AXIS	Invalid minor radius
PGSGCT_E_MINOR_GT_MAJOR	Minor radius is greater than major radius
PGSGCT_E_BAD_LONGITUDE	Invalid longitude
PGSGCT_E_BAD_LATITUDE	Invalid latitude
PGSGCT_E_BAD_DIRECTION	Invalid direction
PGSGCT_E_INVD_SPCS_SPHEROID	Invalid State Plane Coordinates Spheroid (SPCS)
PGSGCT_E_INVD_PROJECTION	Invalid Projection

EXAMPLES: NONE (see example for PGS_GCT_Proj( ))

NOTES: This routine simply initializes the parameters required by a particular projection. The user is referred to the following appendices for further details

Projection List—Appendix G
Parameter List and Use—Appendix G
Spheroid List—Appendix G (State Plane Projection only)

Following steps should be taken if a new projection is to be added to the projection library:

Step 1—archive new code to the projection library

Step 2—define projection code for the new projection in proj.h

Step 3—increment the value of MAXPROJ by one in proj.h

Step 4—add calls to the forward and inverse initialization routines at the end of this file.

Parameters 0 and 1 are reserved for major axis and minor axis respectively

Parameter 4 is reserved for longitude values only.

Parameter 5 is reserved for latitude values only.

Parameters 6 and 7 are reserved for false easting and northing values only.

IMPORTANT All blank array elements are set to zero by the user

Latitude and longitude ranges are as defined in the input section above. The routine checks the longitude value as -PI <= longitude <= PI and latitude as -PI/2 <= latitude <=PI/2. The value of PI is defined as 3.141592653589793238 which is available to the user

State Plane Projection is not available in the Toolkit.

REQUIREMENTS: PGSTK–1500, PGSTK–1502

Transforms Geographical Coordinates into Cartesian Coordinates and Vice Versa for the Given Projection

NAME: **PGS_GCT_Proj( )**

SYNOPSIS:

C:

```
#include <PGS_GCT.h>

PGSt_SMF_status PGS_GCT_Proj(
        PGSt_integer        projId,
        PGSt_integer        directFlag,
        PGSt_integer        nPoints,
        PGSt_double         longitude[],
        PGSt_double         latitude[],
        PGSt_double         mapX[],
        PGSt_double         mapY[],
        PGSt_integer        zone[]);
```

FORTRAN:

```
include"PGS_GCT.f"
include"PGS_GCT_12.f"

integer function pgs_gct_init( projid, directflag, npoints, longitude,
                                        latitude, mapx, mapy, zone)
        integer             projid
        integer             directflag
        integer             npoints
        double precision    longitude(*)
        double precision    latitude(*)
        double precision    mapx(*)
        double precision    mapy(*)
        double precision    zone(*)
```

DESCRIPTION: This tool provides a general interface to perform geo–coordinate transformations in the forward/inverse directions. In general the tool requires a projection id, location of input data vectors and the direction of the conversion. PGSd_UTM projection is a special case for which zone value is also needed for inverse transformations. Forward PGSd_UTM transformations return zone values as output.

INPUTS:

Table 6-254. PGS_GCT_Proj Inputs

Name	Description	Units	Min	Max
projId	projection code	none	1	#defined
directFlag	forward/inverse	none	PGSd_GCT_FORWARD	PGSd_GCT_INVERSE
nPoints	num. of points	none	1	variable
longitude[]	longitude values	radians	-PI	+PI
latitude[]	latitude values	radians	-PI	+PI
mapX[]	x cartesian coordinate (see notes)	meters	variable	variable
mapY[]	y cartesian coordinate (see notes)	meters	variable	variable
zone[]	UTM zones (negative for southern hemisphere)	none	-60	60

OUTPUTS: See description

RETURNS:

Table 6-255. PGS_GCT_Proj Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSGCT_E_BAD_ZONE	Invalid universal transverse mercator (UTM) zone
PGSGCT_E_BAD_DIRECTION	Invalid direction
PGSGCT_E_INVD_PROJECTION	Projection doesn't exists
PGSGCT_E_NO_POINTS	Number of points less than one
PGSGCT_E_GCTP_ERROR	Error in the GCTP library
PGSGCT_E_BAD_LONGITUDE	Bad longitude value (out of range)
PGSGCT_E_BAD_LATITUDE	Bad latitude value (out of range)
PGSGCT_W_INTP_REGION	Interrupted region encountered

EXAMPLES:

C:

```
#include "PGS_GCT.h"

PGS_SMF_status    retValue;
PGSt_double       projParam[15] ;
PGSt_double       latitude[4];
PGSt_double       longitude[4];
PGSt_double       mapX[4], mapY[4];
```

```
PGSt_integer      ProjId = PGSd_UTM;
PGSt_integer      nPoints = 4;
PGSt_integer      directFlag, i;
PGSt_integer      zone[4] = {0, 0, 0, 0};

PGSt_integer      cucFileId;

/* All parameters must be initialized to zero */

for (i = 0; i<15) i++)
{
      projParam[i] = 0;
}

/* C array starts from 0 */

for (i = 1; i<5) i++)
{
      longitude[i-1] = PI/i;
      latitude[i-1] = PI/4;
}
cucFileId = 10999;
retValue =  PGS_CUC_cons(cucFileId,"CLRK80_MAJOR_AXIS",
&ProjParam[0]);
retValue = PGS_CUC_cons(cucFileId,"CLRK80_MINOR_AXIS",
&ProjParam[1]);
      ProjParam[5] = PI/2;
      ProjParam[6] = 3000000 /*(false easting in meters) */
      ProjParam[7] = 75000000 /* (false northing in meters)
*/

      directFlag = PGSd_GCT_FORWARD;
retValue =     PGS_GCT_Init (projId, projParam, directFlag);
retValue =     PGS_GCT_Proj(projId, directflag, nPoints,
                 latitude, longitude, mapX, mapY, zone);

      directflag = PGSd_GCT_INVERSE; (cartesian to
      geographical)
retValue =     PGS_GCT_Init (projId, projParam, directFlag);
retValue =     PGS_GCT_Proj(projId, directflag, nPoints,
                      latitude, longitude, mapX, mapY, zone);
```

FORTRAN:

```
implicit none

include     "PGS_GCT.f"
include     "PGS_GCT_12"
include     "PGS_SMF.f"
```

```
      integer                 PGS_GCT_Proj
      integer                 retValue
      double precision        projParam(15)
      double precision        latitude(4)
      double precision        longitude(4)
      double precision        mapX(4), mapY(4)
      integer                 ProjId
      integer                 nPoints
      integer                 directFlag, i
      integer                 zone(4)

      integer                 cucFileId

      ProjId = PGSd_UTM

      nPoints = 4

C     Projection parameters must be initialized to zero

      do 10 i= 1, 15

            projParam(i) = 0

10    continue

C     FORTRAN array starts from 1

      do 20 i= 1, 4

            longitude(i) = PI/i
            latitude(i) = PI/4

20    continue
      cucFileId = 10999
      pgs_cuc_cons(cucFileId,"CLRK80_MAJOR_AXIS", ProjParam(0));
      pgs_cuc_cons(cucFileId,"CLRK80_MINOR_AXIS", ProjParam(1));
      ProjParam(5) = PI/2

C     false easting in meters
      ProjParam(6) = 3000000

C     false northing in meters
      ProjParam(7) = 75000000

      directFlag = PGSd_GCT_FORWARD
      retValue =     PGS_GCT_Init (projId, projParam, directFlag)
      retValue =     PGS_GCT_Proj(projId, directflag, nPoint
                  latitude, longitude, mapX, mapY, zone)

C     cartesian to geographical
```

```
directflag = PGSd_GCT_INVERSE
pgs_gct_init (projId, projParam, directFlag)
pgs_gct_proj(projId, directflag, nPoints,
             latitude, longitude, mapX, mapY, zone)
```

NOTES: The units of output cartesian coordinates essentially depends on the units used for the Earth's radii, false easting and northing, etc., in the parameters list. The only requirement is that the units used should be consistent.

The zones[] parameter is at present only used for UTM transformations. It's an output parameter in the FORWARD direction and input parameter in the INVERSE direction.

All points are processed even if there is an error condition for some points. If bad point(s) are encountered the routine returns PGSd_GCT_IN_ERROR in the output vector. The user can find out the offending input values by searching for the PGSd_GCT_IN_ERROR in the output vector. For example, if the third point is in error then:

Input Vector	
Longitude	1, 2, 3, 4, 5
latitude	1, 1, 1, 1, 1
Output Vector	
X	.01, .02, PGSd_GCT_IN_ERROR, .04, .05
Y	.1, .2. PGSd_GCT_IN_ERROR, .4, .5

For the inverse transformations, two projections, namely Interrupted Goode and Interrupted Mollweide sometimes encounter a point that is in an interrupted region. In such cases the tool does not abandon processing but puts a value PGSd_GCT_IN_BREAK in the output vector. At the end of processing the tool returns a warning that an Interrupted region was encountered. The user can find out the offending input values by searching for the PGSd_GCT_IN_BREAK in the output vector. For example, if the third point is in the interrupted region:

Input Vector	
X	1, 2, 3, 4, 5
Y	1, 1, 1, 1, 1
Output Vector	
Longitude	.01, .02, PGSd_GCT_IN_BREAK, .04, .05
latitude	.1, .2. PGSd_GCT_IN_BREAK, .4, .5

REQUIREMENTS: PGSTK–1500, 1502

6.3.6 Math and Statistical Support Tools

IMSL has been selected to provide a suite of standard mathematical manipulation functions in a uniform package. This package is available to SCF and DAAC facilities through the EDHS server. Usage of the functionality supplied by the math package will not be mandatory and user developed or shareware routines may be included in science processing software. Users will be responsible for the functionality and long term maintenance of their homegrown software. IMSL service will be provided gratis.

We note that internal Toolkit software does not depend on IMSL.

6.3.7 Constants and Unit Conversions

6.3.7.1 Introduction

The constants and unit conversion tools provide a means to access commonly used mathematical and physical constants, and a coherent means to perform unit conversions and parameter translations.

When the units conversion required is a linear conversion (e.g., degrees to radians) the most efficient mechanism for the programmer is to be given access to a physical constant that describes that transformation and then for the programmer to use this as appropriate. Providing a calling routine to perform the transformation would be inappropriate to most unit conversions and therefore specific API's for this are not provided.

6.3.7.2 Requirements Compliance

a. PGSTK–1521 states that the Toolkit shall provide a means of accessing constant values related to an instrument. Constants that relate to instrument parameters are treated as ancillary data (static internal and dynamic internal) to the algorithms. The mechanism for retrieving instrument ancillary data is described elsewhere in section 6.2.1.6; this requirement is fulfilled by that tool.

b. An external file using some standard parameter=value mechanism will be used to store the mathematical and physical constants, in a similar way as performed in the ancillary data access tools. In this way the values are capable of adjustment without recompilation (requirement PGSTK–1522)

c. PGSTK–1531 states that unit conversion tools shall transform multiple values in a single call. As described above, the most efficient way for the programmer to perform unit conversions is to be given access to the conversion factor, which will be provided by the following tool. This requirement is therefore redundant.

Obtain a Value for a User Specified Constant

NAME: **PGS_CUC_Cons**

SYNOPSIS:

C:

```
#include <PGS_CUC.h>

PGSt_SMF_Status
PGS_CUC_Cons (
        PGSt_integer        inpfileid,
        char                *inpParameter,
        PGSt_double         *outvalue)
```

FORTRAN:

```
include'PGS_CUC_11.f'
include'PGS_SMF_.f'

integer function PGS_CUC_Cons (inpfileid, inpParameter, outvalue)
        integer             inpfileid,
        character           inpParameter,
        double precision    outvalue)
```

DESCRIPTION: This routine receives the fileid and the constant name from the user. The fileid allows more than one input file to be used, thus allowing the user to implement his or her own specialized input files with constants. The parameter is a character string representing the constant whose numerical value is sought by the user. The resulting value is passed back to the user.

INPUTS:

Table 6-256. PGS_CUC_Cons Input

Name	Description	Units	Min	Max
fileid	file identifier	N/A	N/A	N/A
parameter	constant wanted	N/A	N/A	N/A

OUTPUTS:

Table 6-257. PGS_CUC_Cons Output

Name	Description	Units	Min	Max
value	constant value	N/A	N/A	N/A

RETURNS:

Table 6-258. PGS_CUC_Cons Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSCUC_E__ERROR	error in finding the constant value
The following are returned to the error log:	
PGSCUC_E_CANT_GET_FILE_ID	
PGSCUC_E_CANT_OPEN_INPUT_FILE	
PGSCUC_E_AGG_CANT_BE_INSERTED	
PGSCUC_E_READLABEL_PARSE_ERROR	
PGSCUC_E_PARAMETER_INVALID	
PGSCUC_E_FIRST_NODE_NOT_FOUND	

EXAMPLES:

C:

```
char parameter[] = {"PI"};
int  fileid = 19701;
double result;

ret_status = PGS_CUC_Cons(parameter, fileid, result);
```

FORTRAN:

```
implicit none

include     'PGS_CUC_11.f'
include     'PGS_SMF.f'

integer           pgs_CUC_cons
integer           ret_status
integer           inpfileid
character*100     inpParameter
double precision  outvalue
inpfileid = 10790
inpParameter = 'pi'

ret_status = pgs_cuc_cons(inpfileid, inpParameter, outvalue)
ret_value = PGS_CUC_get_parameter(“pi”, pi)
```

NOTES: User defines key word to be searched for within a logical file. User also defines fileid so that location of file can be found. Tool uses ODL libraries to conduct a parameter equals value search. For further information see Constant and Unit Conversions (CUC) Tools Primer, Object Description Language (ODL) documentation.

REQUIREMENTS: PGSTK–1520, PGSTK–1521, PGSTK–1522, PGSTK–1530

Obtain Slope and Intercept to Calculate Conversion Between Specified Units

NAME: **PGS_CUC_Conv**

SYNOPSIS:

C:

```
#include <PGS_CUC.h>

PGSt_SMF_Status
PGS_CUC_Conv (
        char            inpUnit[],
        char            outUnit[],
        PGSt_double     *outSlope,
        PGSt_double     *outIntercept)
```

FORTRAN:

```
include'PGS_CUC_11.f'
include'PGS_SMF.f'

integer function PGS_CUC_Conv(inpUnit, outUnit, outSlope,
                                        outIntercept)
        character*100       inpunit,
        character*100       outunit,
        pgst_double         outslope,
        pgst_double         outintercept)
```

DESCRIPTION: This routine receives two character descriptions of Units as inputs. The first input is the unit that the user has; the second input being the unit the user wants to transform to. Both Unit descriptions are held in a file, after a search identifies whether each unit is held within the file the slope and intercept of the conversion between units is calculated. The resulting values for slope and intercept are then passed back to the user.

INPUTS:

Table 6-259. PGS_CUC_Conv Inputs

Name	Description	Units	Min	Max
inpUnit	unit you have	N/A	N/A	N/A
outUnit	unit you want	N/A	N/A	N/A

OUTPUTS:

Table 6-260. PGS_CUC_Conv Outputs

Name	Description	Units	Min	Max
outSlope	mathematical slope	N/A	N/A	N/A
outIntercept	mathematical intercept	N/A	N/A	N/A

RETURNS:

Table 6-261. PGS_CUC_Conv Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGS_E_CUC_ERROR	error in performing conversion match
The following are returned to the error log:	
PGSCUC_E_COULDNT_INIT_UDUNITS3	
PGSCUC_E_DONT_KNOW_INP_UNIT	
PGSCUC_E_DONT_KNOW_OUTP_UNIT	
PGSCUC_E_UNITS_ARE_INCOMPATIBLE	
PGSCUC_E_A_UNIT_IS_CORRUPTED	

EXAMPLES:

C:

```
char inpUnit[] = {"centigrade"};
char outUnit[] = {"fahrenheit"};
double *outSlope;
double *outIntercept;

ret_status = PGS_CUC_Conv(inpUnit, outUnit, outSlope,
outIntercept);
```

FORTRAN:

```
implicit none

include     'PGS_CUC_11.f'
include     'PGS_SMF.f'

integer           PGS_CUC_Conv
integer           ret_status
character*100     inpUnit
character*100     outUnit
double precision  outSlope
double precision  outIntercept
InpUnit = 'metres'
OutUnit = 'feet'
ret_status = pgs_cuc_conv(inpUnit, outUnit, outSlope,
>                             outIntercept)
```

NOTES: For further details on this tool, see Appendix I. Background on library units used, Units available for conversion and adding own conversion Units to the file.

REQUIREMENTS: PGSTK–1520, PGSTK–1521, PGSTK–1522, PGSTK–1530

6.3.8 Dynamic Memory Management Tools

Allocate Memory

NAME: **PGS_MEM_Malloc( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Malloc(
        void    **addr,
        size_t  numBytes);
```

FORTRAN: None

DESCRIPTION: This tool allocates an arbitrary number of bytes in memory.

INPUTS: numBytes—number of bytes to allocate

OUTPUTS: addr—pointer to beginning of address that has been allocated

RETURNS:

Table 6-262. PGS_MEM_Malloc Returns

Return	Description
PGS_S_SUCCESS	Success
PGSMEM_E_NO_MEMORY	No memory space available for current process
PGSMEM_W_MEMORY_USED	Memory address has been allocated previously

EXAMPLES:

```
int             i;
int             *intPtr = (int *)NULL;
PGSt_SMF_status  returnStatus;

returnStatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10);
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

NOTES: This tool will control the amount of memory that may be allocated at any one time. You should call PGS_MEM_Free( ) to free the memory allocated once you are done using it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process

REQUIREMENTS: PGSTK–1240

NAME: **PGS_MEM_Calloc( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Calloc(
        void    **addr,
        size_t  num_elems,
        size_t  elem_size);
```

FORTRAN: None

DESCRIPTION: This tool allocates an arbitrary number of bytes in memory. All bytes of the allocated memory will be initialized to zero.

INPUTS: num_elems—number of elements

elem_size—size of the element in bytes

OUTPUTS: addr—pointer to beginning address of the memory that has been allocated

RETURNS:

Table 6-263. PGS_MEM_Calloc Returns

Return	Description
PGS_S_SUCCESS	Success
PGSMEM_E_NO_MEMORY	No memory space available for current process
PGSMEM_W_MEMORY_USED	Memory address has been allocated previously

EXAMPLES:

```
int               i;
int               *intPtr = (int *)NULL;
PGSt_SMF_status   returnStatus;

returnStatus = PGS_MEM_Calloc((void
      **)&intPtr,10,sizeof(int));
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

NOTES: This tool will control the amount of memory that may be allocated at any one time. You should call PGS_MEM_Free( ) to free the memory allocated once you are done with it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, that have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

REQUIREMENTS: PGSTK–1240, PGSTK–1241

Re–Allocate Memory

NAME: **PGS_MEM_Realloc( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Realloc(
        void    **addr,
        size_t  newsize);
```

FORTRAN: None

DESCRIPTION: This tool reallocates the number of bytes requested.

INPUTS: addr—pointer to the starting address of previously allocated memory

newsize—new total memory size to reallocate

OUTPUTS: addr—pointer to starting address of newly allocated memory

RETURNS:

Table 6-264. PGS_MEM_Realloc Returns

Return	Description
PGS_S_SUCCESS	Success
PGSMEM_E_NO_MEMORY	No memory space available for current process
PGSMEM_E_ADDR_NOTALLOC	Address is not allocated previously

EXAMPLES:

```
int               i;
int               *intPtr = (int *)NULL;
PGSt_SMF_status   returnStatus;

returnStatus = PGS_MEM_Calloc((void
      **)&intPtr,10,sizeof(int));
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

```
returnStatus = PGS_MEM_Realloc((void
      **)&intPtr,sizeof(int)*20);
if (returnStatus == PGS_S_SUCCESS)
{
   # Realloc success #
}
```

NOTES: This tool will control the amount of memory that needs to be reallocated to a pointer that has already been used to obtain an initial allocation of memory through one of the available Toolkit routines. You should call PGS_MEM_Free( ) to deallocate the memory once you are done using it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, that have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

REQUIREMENTS: PGSTK–1240

Initialize Memory to Zero

NAME: **PGS_MEM_Zero( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_Zero(
        void    *addr,
        size_t  numBytes);
```

FORTRAN: None

DESCRIPTION: This tool initializes a memory block or structure to zero.

INPUTS: addr—beginning address of the memory block or structure

numbytes—number of bytes

OUTPUTS: None

RETURNS: None

EXAMPLES:

```
Typedef     struct
{
    int     i;
    char    c;
    float   f;
}TestStruct;

TestStruct        test
int               *intptr = (int *)NULL
returnstatus      returnstatus

PGS_MEM_Zero(&test,sizeof(test));
returnstatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10 );
if (returnstatus == PGS_S_SUCCESS)
{
    PGS_MEM_Zero(intPtr,sizeof(intPtr)*10)
}

PGS_MEM_Zero( s, sizeof(longint)*10 );
```

REQUIREMENTS: PGSTK–1240

De–Allocate Memory

NAME: **PGS_MEM_Free( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_Free(
        void    *addr);
```

FORTRAN: None

DESCRIPTION: This tool deallocates memory that was previously allocated through the use of a Toolkit allocation routine.

INPUTS: addr—address of previously allocated memory

OUTPUTS: None

RETURNS: None

EXAMPLES:

```
int               *intPtr = (int *)NULL;
returnStatus      returnStatus;

returnStatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10);
if (returnStatus == PGS_S_SUCCESS)
{
    PGS_MEM_Free(intPtr);
    intPtr = (int *)NULL;
}
```

NOTE: Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

REQUIREMENTS: PGSTK–1240

NAME: **PGS_MEM_FreeAll( )**

SYNOPSIS:

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_FreeAll(
        void);
```

FORTRAN: None

DESCRIPTION: Deallocates all memory that was previously allocated through the use of Toolkit allocation routines, within an executable. Calls to PGS_MEM_Free( ) and PGS_MEM_FreeAll( ) may be interlaced.

INPUTS: None

OUTPUTS: None

RETURNS: None

EXAMPLES:

```
typedef struct
{
   int      i;
   char     c;
   float    f;
}TestStruct;

TestStruct  *test       = (TestStruct *)NULL;
int         *intPtr     = (int *)NULL;

PGS_MEM_Malloc((void **)&intPtr,sizeof(int)*10);
PGS_MEM_Malloc((void **)&test,sizeof(TestStruct)*10);
PGS_MEM_FreeAll();

test        = (TestStruct *)NULL
intPtr      = (int *)NULL
```

NOTES: This tool should only be called near the end of processing, or when no further allocation of dynamic memory will be required.

Due to the comprehensive nature of this tool, all allocated memory references, that have not yet been freed, will be disposed of. Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to

use. Failure to heed these warnings may result in anomalous behavior within your process.

REQUIREMENTS: PGSTK–1240

6.3.9 Graphics Support Tools

These tools will support the analysis of graphics, quicklook and quality assurance (QA) data output from science production processes. It is assumed that the exchange format of the data files will be HDF, although specific graphics formats are TBD at the time of this document. These tools will contain an image processing capability, which will be used in conjunction with the math library chosen for the Toolkit (section 6.3.6), or with user supplied math functions. It is expected that this functionality will be a subset of the data visualization capability supplied by EOSVIEW, a package being developed to display EOS data structure.

This page intentionally left blank.

Appendix A. Assumptions

The following is a list of assumptions made in developing the specification of the routines in the SDP Toolkit described in section 6.

A.1 SDP Toolkit Tools—Mandatory

A.1.1 File I/O Tools

A.1.1.1 Level 0 Science Data Access Tools

PGS_IO_L0_Open( )

PGS_IO_L0_GetHeader( )

PGS_IO_L0_GetPacket( )

a. Level 0 raw data will be in the form of CCSDS–formatted packets.

b. Level 0 packets will be time–ordered and duplicate packets will have been removed by EDOS or Pacor/DDF.

c. Level 0 access routines are designed to operate on physical files, which may not be identical to data granules.

d. Level 0 data files, with associated file attribute metadata, will come through the Science Data Processing Segment (SDPS) ingest data server and will be pre–staged to a given PGE.

e. ECS Data Ingest will stage and make available file attribute metadata for each physical Level 0 data file staged to a PGE.

f. Without changing any physical file data, ECS Data Ingest will perform any granularization of Level 0 data to a form other than as is received from SDPF or EDOS (if this does not correspond to the form required by EOS investigators) prior to the staging of Level 0 data to the PGE.

g. ECS Data Ingest will perform any EOS investigator required subsetting or combination of Level 0 header and quality information that is necessary as a result of granularizing Level 0 data files prior to the staging of the data to the PGE.

h. ECS Data Ingest will make information on the orbit number corresponding to each physical Level 0 data file available to the SDP Toolkit through associated metadata.

i. For each SDPF–generated Data Set File staged to a PGE by ECS Data Ingest, the corresponding SFDU header file will also be staged.

j. Level 0 data files will be staged to a PGE in the machine native format.

k. For each staged Level 0 data file, the following file attribute metadata parameters, at a minimum, will be staged and available to a PGE for use in science processing:

1. time tag of 1st packet of staged Level 0
2. time tag of last packet of staged Level 0
3. number of physical Level 0 data files staged
4. start time of Level 0 data as requested by investigators through the planner/scheduler system
5. end time of Level 0 data as requested by investigators through the planner/scheduler system
6. APID of each Level 0 data file, if the Level 0 data files are APID–unique
7. orbit number(s) of the staged Level 0 data

A.1.1.2 HDF File Access Tools

a. It is assumed that users will obtain and compile the HDF libraries of the HDF Group on their own and link with the PGS. (HDF4 and HDF5 distribution is available via http://www.hdfgroup.org/.)

A.1.1.4 Metadata

PGS_MET_Init( )

a. A Metadata Configuration File (MCF) will be built around the 'parameter = value' form to provide maximum flexibility. Each metadata element will be fully described in the MCF. This information will be held in memory in a set of linked structures or similar constructs.

b. The core metadata descriptions will be supplied by ECS.

c. It is assumed that only one header will be initiated at any one time during processing.

PGS_MET_Write( )

a. It is assumed that the output of the metadata tools will be to an HDF (HDF4 or HDF5) formatted product. In each case the product/file may be existing or new. It is assumed that these products/files will be opened and closed using the appropriate tools (e.g., open/close generic file); i.e., the _MET_ tools do not perform these functions.

b. It is assumed that further interaction with the inventory is done using other software that interacts with the metadata file produced by this tool.

PGS_MET_GetPCAttr( )

a. It is assumed that input products are accessed through the PCF and associated tools

b. It is assumed that the metadata in input files is available either 1. in the same form as that written by PGS_MET_Write or 2. in a simple separate ASCII text file. In both cases, the metadata file is referenced in the field prescribed by the PCF rules.

PGS_MET_GetConfig ( )

a. It is assumed that configuration data is held as prescribed by the PCF rules.

b. It is assumed that configuration data will be accessed using the label field.

A.1.2 Error/Status Reporting Tools

a. It is assumed that only three log files will need to be created by the Toolkit: Status Message Log, User Status Log and Status Report Log.

b. Every call to a PGS_SMF_Set* routine results in a status message being appended to the Status Log file.

c. Status Report entries are directed to the Status Report Log file.

d. User Status entries are directed to the User Status Log file.

PGS_SMF_SetHDFMsg( )

a. It is assumed that calls to HDF–EOS library routines will set or return an error code and message that can be retrieved by this function for later recall by other error reporting tools, or that the HDF–EOS library will incorporate the existing SMF library calls thereby circumventing the need for this tool.

PGS_SMF_GetActionByCode( )

a. It is assumed that the user only requires the specification and retrieval of an action string, for use in reporting, and not the specification and execution of action methods.

PGS_SMF_CreateMsgTag( )

a. Assumption is that this tool will have access to production run id and science software program id during runtime; thus enabling this routine to generate a unique string based on product id.

PGS_SMF_GenerateStatusReport( )

a. It is assumed that the Toolkit development team has the license to determine the format of the individual status report entries. The format that we have adopted calls for a system–defined message tag to precede a user–provided message string; separators will be inserted between individual report entries for the sake of clarity.

b. It is assumed that the generation of a status report results in the report being entered into a Status Report Log file created by the Toolkit.

PGS_SMF_SendRuntimeData( )

a. It is assumed that this toolkit will interface to some other toolkit, or Communications and Systems Management Segment (CSMS) functionality, to effect the transfer of the selected Runtime files to an intermediate holding location. The same mechanism will perform the transmission of one or more e–mail notices to alert the interested parties as to the disposition of the Runtime files.

b. It is also assumed that there will be a defined intermediate holding location for this toolkit to send the Runtime files at the DAAC site and that there will be an interface to alert the monitoring authority that these Runtime files have arrived.

A.1.3 Process Control Tools

a. a PGE process control database record will exist as a UNIX file or Database Management System (DBMS) record for each PGE within the DAAC.

b. A template PGE process control database record will be "seeded" with user–defined information during the integration and testing process.

c. An instance of the PGE process control database record will be populated with the appropriate runtime data and if necessary, staged prior to PGE execution.

d. Runtime parameter values may be modified prior to runtime through some as yet unidentified interface/mechanism.

e. A one–to–many logical–to–physical file relationship may exist for input product, output product, input support and output support files.

f. The Planning & Data Production System (PDPS) will provide for Toolkit initialization allowing internal Toolkit structures to become populated.

g. The PDPS will provide for Toolkit termination, allowing the Toolkit to perform necessary housekeeping and ensuring that important intermediate data gets saved for future runs of the same PGE.

PGS_PC_GenUniqueID( )

a. It is assumed that the Science Software Program ID and the Production Run ID are system defined values that will be available from the execution environment, or from the PGE process control database during Toolkit Initialization.

b. The logical Product ID value passed in by the user will be defined by the user, but will have been mapped to a DAAC–based intermediate identifier during the Integration & Test phase.

PGS_PC_GetConfigData( )
PGS_PC_GetConfigDataCom

a. Each user–defined logical Runtime Parameter ID passed into this function will be mapped to an actual runtime parameter during I&T. This will allow the Parameter ID to be resolved into a default value, or an overriding value at runtime.

PGS_PC_GetReference( )

a. It is assumed that users of HDF will utilize this tool to obtain a reference to pass to the HDF open library call.

PGS_PC_GetNumberOfFiles( )
PGS_PC_GetNumberOfFilesCom

a. To satisfy the one–to–many logical–to–physical file relationship, the user, upon retrieving the number of files per given identifier with this tool, will be able to index to the desired instance of a file by providing the version number to the appropriate file I/O toolkit function.

PGS_PC_GetFileAttr( )
PGS_PC_GetFileByAttr( )
PGS_PC_GetFileAttrCom

a. It is assumed that input product metadata and file attributes will be made directly available to the Toolkit through the PGE Process Control Database.

b. If available, it is assumed that input support file metadata and file attributes will be made directly available to the Toolkit through the PGE Process Control Database.

A.1.4 Memory Management Tools

Dynamic Memory Tools

a. It is assumed that all dynamic memory allocated within the user's program is obtained through the use of these tools.

Shared Memory Tools

a. One basic assumption is that all the executables will be invoked within a shell script (i.e., PGE).

b. Additionally, that there will be a shell script that wraps around the main PGE shell script, allowing an initialization program to create a shared memory segment for the Toolkit; this will enable the Toolkit to facilitate tracking of all the necessary resources needed to support shared memory capabilities for the user. That same shell script will allow a termination program to release all the shared–memory resources used by both the Toolkit and the user.

c. Modification to the existing shared memory API will be minimal if and when the POSIX implementation is adopted.

d. Shared memory segments will be large enough to support the needs of both the user and the Toolkit.

e. Two segments, one for the user and one for the Toolkit, can be attached concurrently within the same process.

A.1.5 Bit Manipulation Tools

a. It is assumed that bit–manipulation functionality will be provided inherently by the language for 'C' and Fortran90, and that users of Fortran77 will use compilers that conform to MIL STD 1753 in order to obtain these capabilities.

A.1.6 Spacecraft Ephemeris and Attitude Data Access Tools

PGS_EPH_EphemAttit( )

a. The specification for reliability of orbit and attitude data is assumed to be provided by Goddard Space Flight Center (GSFC)/Flight Dynamics Facility (FDF).

b. This tool does not compute instrument attitude.

c. Time is assumed to be input in ASCII time code A or B format.

A.1.7 Time and Date Conversion Tools

PGS_TD_UTCtoTAI( )

a. The current leap seconds file must be available.

PGS_TD_TAItoUTC( )

a. The current leap seconds file must be available.

PGS_TD_UTCtoGPS( )

a. The current leap seconds file must be available.

PGS_TD_GPStoUTC( )

a. The current leap seconds file must be available.

PGS_TD_SCtime_to_UTC( )

a. The Spacecraft time difference file or coefficients for interpolation must be available. The current leap seconds file must be available.

PGS_TD_UTC_to_SCtime( )

a. The Spacecraft time difference file or coefficients for interpolation must be available. The current leap seconds file must be available. User responsibility to work with difference from nearest tick (interpolate between ticks if desired). It is assumed that this requirement is intended for cross checking of data and that the usual transformation is from Spacecraft Clock time to other standards, such as UTC. If the user wants to interpolate, they will

have to take answer back to UTC and find the difference from the original UTC; then go to next tick on that side and interpolate between the two. It would be possible to rework this tool to provide the two nearest ticks on either side of the UTC time and interpolation weights.

PGS_TD_TimeInterval( )

a. It is user responsibility to supply TAI times, although GPS times can be used instead. The two must not be mixed. All the function does is to subtract double precision numbers.

A.2 SDP Toolkit Tools - Optional

A.2.2 Ancillary Data Access and Manipulation Tools

PGS_AA_dcw( )

a. It is assumed that for access to areas or multiple points, that the user will provide the lat/long coordinates to this tool; i.e., the tool does not include the functionality to calculate other coordinates than those supplied by the user.

PGS_AA_dem( )

a. It is assumed that DEMs will be in raster format.

b. All assumptions under **PGS_AA_2DRead( ) and PGS_AA_2Dgeo( ) apply.**

PGS_AA_PeVA( )

a. It is assumed that a large number of static files holding data associated with various algorithms will be in ASCII format. It is further assumed that some of these files will be in the parameter = value format.

PGS_AA_2DRead( ) and PGS_AA_2Dgeo( )

a. It is assumed that the ancillary data have been prepared into formats suitable for use with this tool; i.e., they are in 2D grids containing data values organized in a raster format and describable using a standard set of metadata.

b. It is assumed that the ancillary data files will exist as a series of time specific physical files with a clear time–tag (e.g., in the file name); i.e., each physical file contains a full set of the data in spatial terms (e.g., sea ice for one week for the region north of 60 degrees).

c. It is assumed that for most purposes, a 2 dimensional array of sufficient size can be created to service user requirements.

PGS_AA_3DRead( ) and PGS_AA_3Dgeo( )

a. It is assumed that the ancillary data have been prepared into formats suitable for use with this tool; i.e., they are in 3D grids containing data values organized in a raster format and describable using a standard set of metadata.

b. It is assumed that the ancillary data files will exist as a series of time specific physical files with a clear time–tag (e.g., in the file name); i.e., each physical file contains a full set of the data in spatial terms.

c. It is assumed that for most purposes, a 3 dimensional array of sufficient size can be created to service user requirements.

PGS_AA_INTERP( )

This functionality is now part of PGS_AA_2Dgeo. See section D.3.2.3

A.2.3 Celestial Body Position

A.2.3.1 Celestial Body Access Tools

PGS_CBP_Earth_CB_Vector( )

a. Sun, moon, and planetary ephemerides are assumed to exist in an external file.

b. Time is assumed to be input in CCSDS ASCII time code A or B format.

PGS_CBP_Sat_CB_Vector( )

a. Sun, moon, and planetary ephemerides are assumed to exist in an external file.

b. Time is assumed to be input in CCSDS ASCII time code A or B format.

c. Spacecraft ephemeris is assumed to be available in an external file.

d. Earth to Celestial Body ECI vector is assumed to be computed using the tool of that name.

PGS_CBP_SolarTimeCoords( )

a. Time is assumed to be input in ASCII time code A or B format.

PGS_CBP_body_inFOV( )

a. Sun, moon, and planetary ephemerides are assumed to exist in an external file.

b. Star locations are assumed to be read from the mission star catalog file received from FDF.

c. A set of vectors defining the FOV in spacecraft coordinates is assumed to be provided by the user. The vectors must be in sequential order around the FOV periphery.

d. Time is assumed to be input in ASCII time code A or B format.

e. Spacecraft ephemeris is assumed to be available in an external file.

PGS_CBP_BrightStar_positions( )

a. Star locations are assumed to be read from the mission star catalog file.

b. The star catalog is assumed to be created based on a minimum star magnitude TBD by the project.

c. Time is assumed to be input in ASCII time code A or B format.

A.2.4 Coordinate System Conversion

A.2.4.1 Coordinate System Conversion - Transformation Tools

A.2.4.2 Coordinate System Conversion - Other Tools

PGS_CSC_DayNight( )

a. The position of the sun is assumed to be obtained from the sun, moon, and planetary ephemerides external file.

b. Time is assumed to be input in CCSDS ASCII time code A or B format.

PGS_CSC_GreenwichHour( )

a. A file of UT1–UTC times is assumed to be present.

b. Time is assumed to be input in CCSDS ASCII time code A or B format.

PGS_CSC_SubSatPoint( )

a. Time is assumed to be input in CCSDS ASCII time code A or B format.

b. Spacecraft ephemeris is assumed to be available in an external file.

c. Earth oblateness model is assumed to be the same as that used to compute the spacecraft ephemeris originally.

d. A file of UT1–UTC times and Earth polar motion is assumed to be present.

PGS_CSC_Earthpt_FOV( )

a. A set of vectors defining the FOV in spacecraft coordinates is assumed to be provided by the user. The vectors must be in sequential order around the FOV periphery.

b. Time is assumed to be input in CCSDS ASCII time code A or B format.

c. Spacecraft ephemeris is assumed to be available in an external file.

d. Earth oblateness model is assumed to be the same as that used to compute the spacecraft ephemeris originally.

e. User must supply one vector inside FOV—preferably near center

A.2.5 Geo–Coordinate Transformation Tools

a. It is assumed that the user has knowledge of the values of the necessary initialization parameters or uses those from the CUC tools (where available).

A.2.6 Constants and Unit Conversions

a. It is assumed that the constants in this section are supplied by ESDIS.

Appendix B. Status Message File (SMF) Creation and Usage Guidelines

B.1 Note

For a much more simplified explanation about SMF Creation and Usage Guidelines, refer to the SDP Toolkit Primer. The Primer is available on the World Wide Web (WWW). The Universal Research Locator (URL) for the ECS Data Handling System (EDHS) home page is:

http://edhs1.gsfc.nasa.gov/

This appendix provides a more detailed description of how Status Message Files (SMFs) are created along with some guidelines on their usage within the science software. Additionally, some examples are provided at the end of this appendix to better illustrate how the software may be used.

B.2 Description

In EOS, messages to the user should be developed using the Status Message File tool set. Together, these tools provide the means to store messages in files that are accessed at runtime to retrieve context–specific message text. Since text messages are stored in runtime files, messages may be modified without recompiling the program that uses the messages. The basic procedure for using these tools follows:

- Create a Status Message File (SMF) that maps status message text to a status label. Additionally, the user may create action message text which maps to the same status label, though this is optional.
- Compile the SMF using the 'smfcompile' program to generate the runtime message file and language–specific "include" file. The runtime message file is used to hold the message/action text. The language–specific "include" file maps the status labels to numeric status numbers via language–specific constructs.
- Use PGS_SMF_Set* tools to preserve a specific status condition.
- Use PGS_SMF_Get* tools to retrieve messages/actions based on the status labels returned by previously called functions.

  SMFs require a seed number that is used to generate message/action numbers for message/action labels. This seed number is the key to determining the proper runtime message file and must be unique for each message file. Users cannot simply use any seed number they wish to; they have to be requested and/or assigned by the PGS Toolkit development team. Currently we can support seed numbers up to (2^19)-1 (i.e., 524287). To help identify the proper runtime message file, all message files will be located in a

common message directory, located by the environment variable PGSMSG. This directory will be created by the Toolkit install facility and updated during an smf make procedure.

New updates to this directory may be performed by compiling an SMF text file in the message directory. A more advisable approach would be to maintain each SMF text file in the same directory as the code that relies on the messages contained in the SMF text file. Then compilation of the SMF text file(s) could be setup to precede compilation of the source code (e.g., make smf; make code).

Status Message text file names can be of any valid UNIX filename characters; they must however include a '.t' extension. The generated runtime ASCII message file will be named as PGS_<seed#>, (e.g., PGS_255). The resulting "include" file follows the convention PGS_<tool–group>_<seed#>.[haf] (e.g., PGS_IO_1.h & PGS_IO_1.f). The token <tool–group> is extracted from the 'LABEL' field contained in the SMF text file. For this reason, it would be advisable to name SMF text files with some portion of this field in order to maintain some relationship between the original text file and the smf generated files. To provide a consistent method of status returns, the following procedures should be followed for all software developed for EOS:

- All functions should return one of the following return codes as defined in PGS_SMF.h (FORTRAN users refer to PGS_SMF.f) to indicate the status of the Toolkit operation, unless the function returns a user–defined status as defined in an SMF, or unless a return is unwarranted altogether as in a simple mathematical function (e.g., y = sine(x)):

PGS_S_SUCCESS	Successful operation
PGS_E_ECS	A general ECS error occurred
PGS_E_TOOLKIT	A general TOOLKIT error occurred
PGS_E_UNIX	A UNIX error occurred
PGS_E_HDF	An HDF–EOS error occurred
PGS_E_DCE	A DCE error occurred
PGS_E_ENV	A Toolkit environment error was detected

  Note that additional defined return codes will be added for various COTS/modules in the future should the need arise.

- Before returning a status code, the unit (i.e., routine, function, procedure, etc.,) should load the specific status information into the static buffer. This is accomplished by calling one of the PGS_SMF_Set* tools.

- The calling function should check the return status of the called unit. If an error condition occurred, the specific error data can be retrieved using the PGS_SMF_Get* tools.

The tools that set or retrieve status data to/from the static buffer area are listed under PGS Error/Status Reporting Tools in the Toolkit User's Guide.

SMF syntax: Syntax for SMF definition is specified in the variant Backus–Naur Form (BNF) notation that follows:

BNF notes : [optional item]; { range bounded}; + concatenation [ ] and space symbols indicate blank or space character

```
allowed_ascii_char   ::= {   [! " # & ' ( ) % * + , - . /]
                             [DIGIT]
                             [: ; < = > ? @]
                             [UPPER_CASE_LETTER]
                             [LOWER_CASE_LETTER]
                             [[ \ ] ^ _ ` { | } ~] }
spacing              ::= {[\n] [\t] [ ]}
comment_str          ::= #
instrument           ::= 3{[UPPER_CASE_LETTER]}10
label                ::= 3{[UPPER_CASE_LETTER]}10
level                ::= S | M | U | N | W | E | F
mnemonic             ::=1{[DIGIT][_][UPPER_CASE_LETTER]}31
mnemonic_label       ::= label + _ + level + _  + mnemonic
action_label         ::= label + _ + A + _ + mnemonic
message_str          ::= 1{[ ] [allowed_ascii_char]}240
action_str           ::= message_str
status_definition    ::= mnemonic_label + spacing +
                             message_str
                             [+ :: + action_label]
action_definition    ::= action_label + spacing + action_str
```

Note on levels:

_S_	stands for success
_A_	stands for action (action_label definition only)
_M_	stands for message
_U_	stands for user information
_N_	stands for notice
_W_	stands for warning
_E_	stands for error
_F_	stands for fatal

It is up to the user to use the appropriate level in their definition of mnemonics that represent message/action strings. So if an action string is required, use the _A_ sequence in the action_label; if it is an informational–message string use the _M_ sequence in the mnemonic_label; if it is a fatal message string use _F_ in the mnemonic_label. Only action_labels use an action level character; the rest of your mnemonic_label definitions should use other level characters.

This page intentionally left blank.

Appendix C. Process Control Files

NOTE:

The Master Template PCF as delivered with the Toolkit and described in section C.1.4, MUST be used in its entirety as a template for user PCFs. Please add to it, but do not alter any entries now in it. This file has been populated with dependency information required for proper operation of the Toolkit.

For a much more simplified explanation about Process Control Files and usage, refer to the SDP Toolkit Primer. The Primer is available on the World Wide Web (WWW) http://newsroom.gsfc.nasa.gov/sdptoolkit/primer/tkprimer.html. The Universal Reference Locator (URL) for the ECS Data Handling System (EDHS) home page is:

http://edhs1.gsfc.nasa.gov

This appendix provides a detailed description of how to define and validate Process Control Files.

C.1 Defining Process Control Files

This section of the appendix discusses the various components of a Process Control File (PCF). A sample PCF format is provided as well as an example, which contains the actual entries required to support the Toolkit release 5.2.20

C.1.1 PCF Components

- **Subject Fields** A process control file MUST contain the following subject fields in the order shown:

System Runtime Parameters	- unique identifiers used to track instances of a PGE run, versions of science software, etc.
Product Input Files	- list of ECS standard product data files required as input to the PGE
Product Output Files	- list of ECS standard product data files generated by the PGE
Support Input Files	- list of ECS, or Instrument ancillary/support data files required as input to the PGE
Support Output Files	- list of ECS, or Instrument ancillary/support data files generated by the PGE
User Defined Runtime	- list of user–defined configuration parameters; Parameters to be accessed by the PGE at runtime

Intermediate Input	-	list of non–volatile temporary files required as input to the PGE
Intermediate Output	-	list of non–volatile temporary files generated by the PGE
Temporary I/O	-	list of volatile temporary files generated and accessed by the PGE at runtime only
End	-	PCF terminus

- **Record Fields** Each dependency record MUST contain, in the proper order, all of the fields required for the particular type of Subject.

Identifier	-	Numeric representation of logical identifier (range 10,000–10,999 reserved for Toolkit use only)
Reference	-	UNIX file/directory name
Path	-	UNIX directory path; start paths with '~' to specify relative paths from $PGSHOME
Reserved	-	Placeholder for future use
Universal	-	Universal Reference identifier - may be any string and may contain spaces
Attribute	-	Full UNIX path to Product Attribute file
Sequence	-	Number of associated Product Input files to follow (inclusive); typically = 1
Description	-	Annotation for parameter; not used in processing
Value	-	Assignment to be used during processing; string representation returned by tools

C.1.2 Format Rules

- All Subject fields are placed in the order shown above
- Each subject field must begin with the question mark token '**?**'
- The default location entry, for a subject field, must begin with the bang token '!'; there may be only one such entry per subject field and it must immediately follow the subject field declaration.
- All comments must begin with the pound sign token '**#**'
- Subject and comment tokens must be placed in column one
- There can be no blank lines in the file

- All Record entries must begin in column one
- All Record fields must be delimited with a pipe token '|'
- The last line of the file must begin with a subject field token '**?**'

C.1.3 Format Example

```
# Process Control Information File
#
#     The Environment variable PGS_PC_INFO_FILE must point to this file.
#     Required inputs appear in bold; all delimiters required.
#     'Path' obtained
#     from the default location entry unless explicitly defined for the
#     individual record.
#
?   SYSTEM RUNTIME PARAMETERS
# ------------------------------------------------------------------------
# Production Run ID - unique production instance identifier (SCF=1)
# ------------------------------------------------------------------------
Value
# ------------------------------------------------------------------------
# Software ID - unique software configuration identifier        (SCF=1)
# ------------------------------------------------------------------------
Value
#
?   PRODUCT INPUT FILES
! ~/runtime
#
# ------------------------------------------------------------------------
# Sequence number must be ordered in a descending fashion
# Ex.
# 100|Instr_Product1A_1.dat|/usr/data||Product1A 1|/usr/data/prod_1A_1.att|3
# 100|Instr_Product1A_2.dat|/usr/data||Product1A 2|/usr/data/prod_1A_2.att|2
# 100|Instr_Product1A_3.dat|/usr/data||Product1A 3|/usr/data/prod_1A_3.att|1
#
# Attribute file MUST reside in same directory as Reference file
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Attribute|Sequence
#
?   PRODUCT OUTPUT FILES
! ~/runtime
#
```

```
# ------------------------------------------------------------------------
# Sequence number must be ordered in a descending fashion
# Attribute file MUST reside in same directory as Reference file
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Attribute|Sequence
#
?   SUPPORT INPUT FILES
! ~/runtime
#
# ------------------------------------------------------------------------
# Sequence number = 1;
# Attribute file MUST reside in same directory as Reference file
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Attribute|Sequence
#
?   SUPPORT OUTPUT FILES
! ~/runtime
#
# ------------------------------------------------------------------------
# Sequence number = 1;
# Attribute file MUST reside in same directory as Reference file
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Attribute|Sequence
#
?   USER DEFINED RUNTIME PARAMETERS
#
# ------------------------------------------------------------------------
# Value may contain white-space but must be limited to current line;
# Value returned by Toolkit in string representation
# ------------------------------------------------------------------------
Identifier|Description|Value
#
?   INTERMEDIATE INPUT
! ~/runtime
#
# ------------------------------------------------------------------------
# Sequence number = 1;
# Records obtained from INTERMEDIATE OUTPUT field of previous runs
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Reserved|Sequence
#
?   INTERMEDIATE OUTPUT
! ~/runtime
#
```

```
# ------------------------------------------------------------------------
# Sequence number = 1;
# Records generated by Toolkit ONLY!
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Universal|Reserved|Sequence
#
?   TEMPORARY I/O
! ~/runtime
#
# ------------------------------------------------------------------------
# Sequence number = 1;
# Records generated by Toolkit ONLY!
# ------------------------------------------------------------------------
Identifier|Reference|Path|Reserved|Reserved|Reserved|Sequence
#
?   END
```

C.1.4 Master Template:

The following file was delivered along with the Toolkit Installation. To access this file, set the environment variable PGS_PC_INFO_FILE to '$PGSHOME/runtime/PCF.relA'.

Initially, this file has been populated with dependency information required for proper operation of the Toolkit. As such, **this file should be considered as a MASTER PCF file from which user PCF files are derived.** To safeguard against the possibility of corrupting essential Toolkit entries, users should use copies of this file as the basis for creating their own. Once a new PCF file has been created, reset the environment variable PGS_PC_INFO_FILE to point to the new file. The new file should now contain all the essential User and Toolkit dependency information. Before using the new PCF, please validate it using the 'pccheck.sh' utility that is located in $PGSHOME/bin. The effort spent doing so will more than offset the time spent trying to debug the PCF from the errors received while running your program(s). Refer to Part II of this Appendix to see an example on the usage of the 'pccheck.sh' PCF validation tool.

```
#

# filename:

#     PCF.relB0

#

# description:

#     Process Control File (PCF)

#

# notes:

#
```

```
#      This file supports the Release B version of the toolkit.
#       It is intended for use with toolkit version "TK_VERSION_STRING".
#
#       The logical IDs 10000-10999 (inclusive) are reserved for internal
#       Toolkit/ECS usage, DO NOT add logical IDs with these values.
#
#      Please treat this file as a master template and make copies of it
#      for your own testing. Note that the Toolkit installation script
#      sets PGS_PC_INFO_FILE to point to this master file by default.
#       Remember to reset the environment variable PGS_PC_INFO_FILE to
#       point to the instance of your PCF.
#
#       The toolkit will not interpret environment variables specified
#       in this file (e.g. ~/database/$OSTYPE/TD is not a valid reference).
#       The '~' character, however, when appearing in a reference WILL be
#       replaced with the value of the environment variable PGSHOME.
#
#       The PCF file delivered with the toolkit should be taken as a
#       template.  User entries should be added as necessary to this
#       template.  Existing entries may (in some cases should) be altered
#       but generally should not be commented out or deleted.  A few
#       entries may not be needed by all users and can in some cases
#       be commented out or deleted.  Such entries should be clearly
#       identified in the comment(s) preceding the entry/entries.
#
#       Entries preceded by the comment: (DO NOT REMOVE THIS ENTRY)
#       are deemed especially critical and should not be removed for
#       any reason (although the values of the various fields of such an
#       entry may be configurable).
#
```

```
# ------------------------------------------------------------------------
?   SYSTEM RUNTIME PARAMETERS
# ------------------------------------------------------------------------
##########################################################################
#
# This section contains unique identifiers used to track instances of
# a PGE run, versions of science software, etc.  This section must
# contain exactly two entries.  These values will be inserted by
# ECS just before a PGE is executed.  At the SCF the values may be set
# to anything but these values are not normally user definable and user
# values will be ignored/overwritten at the DAAC.
#
##########################################################################
#
# Production Run ID - unique production instance identifier
# (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
1
# ------------------------------------------------------------------------
# Software ID - unique software configuration identifier
# (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
1
#
?   PRODUCT INPUT FILES
##########################################################################
#
# This section is intended for standard product inputs, i.e., major
# input files such as Level 0 data files.
#
```

```
# Each logical ID may have several file instances, as given by the
# version number in the last field.
#
################################################################################
#
# Next non-comment line is the default location for PRODUCT INPUT FILES
# WARNING! DO NOT MODIFY THIS LINE unless you have relocated these
# data set files to the location specified by the new setting.
!  ~/runtime
#
# ------------------------------------------------------------------------
# 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.  These entries may be removed IF the AA tools
# are not being used.
# ------------------------------------------------------------------------
10780|usatile12|AA_DATA_INSTALL_DIR|||10751|12
10780|usatile11|AA_DATA_INSTALL_DIR|||10750|11
10780|usatile10|AA_DATA_INSTALL_DIR|||10749|10
10780|usatile9|AA_DATA_INSTALL_DIR|||10748|9
10780|usatile8|AA_DATA_INSTALL_DIR|||10747|8
10780|usatile7|AA_DATA_INSTALL_DIR|||10746|7
10780|usatile6|AA_DATA_INSTALL_DIR|||10745|6
10780|usatile5|AA_DATA_INSTALL_DIR|||10744|5
10780|usatile4|AA_DATA_INSTALL_DIR|||10743|4
10780|usatile3|AA_DATA_INSTALL_DIR|||10742|3
10780|usatile2|AA_DATA_INSTALL_DIR|||10741|2
10780|usatile1|AA_DATA_INSTALL_DIR|||10740|1
10951|mowe13a.img|AA_DATA_INSTALL_DIR||||1
10952|owe13a.img|AA_DATA_INSTALL_DIR||||1
```

```
10953|owe14d.img|AA_DATA_INSTALL_DIR||||1
10954|owe14dr.img|AA_DATA_INSTALL_DIR||||1
10955|etop05.dat|AA_DATA_INSTALL_DIR||||1
10956|fnocazm.img|AA_DATA_INSTALL_DIR||||1
10957|fnococm.img|AA_DATA_INSTALL_DIR||||1
10958|fnocpt.img|AA_DATA_INSTALL_DIR||||1
10959|fnocrdg.img|AA_DATA_INSTALL_DIR||||1
10960|fnocst.img|AA_DATA_INSTALL_DIR||||1
10961|fnocurb.img|AA_DATA_INSTALL_DIR||||1
10962|fnocwat.img|AA_DATA_INSTALL_DIR||||1
10963|fnocmax.imgs|AA_DATA_INSTALL_DIR||||1
10964|fnocmin.imgs|AA_DATA_INSTALL_DIR||||1
10965|fnocmod.imgs|AA_DATA_INSTALL_DIR||||1
10966|srzarea.img|AA_DATA_INSTALL_DIR||||1
10967|srzcode.img|AA_DATA_INSTALL_DIR||||1
10968|srzphas.img|AA_DATA_INSTALL_DIR||||1
10969|srzslop.img|AA_DATA_INSTALL_DIR||||1
10970|srzsoil.img|AA_DATA_INSTALL_DIR||||1
10971|srztext.img|AA_DATA_INSTALL_DIR||||1
10972|nmcRucPotPres.datrepack|AA_DATA_INSTALL_DIR||||1
10973|tbase.bin|AA_DATA_INSTALL_DIR|||10915|1
10974|tbase.br|AA_DATA_INSTALL_DIR|||10919|4
10974|tbase.bl|AA_DATA_INSTALL_DIR|||10918|3
10974|tbase.tr|AA_DATA_INSTALL_DIR|||10917|2
10974|tbase.tl|AA_DATA_INSTALL_DIR|||10916|1
10975|geoid.dat|AA_DATA_INSTALL_DIR||||1
#
# ------------------------------------------------------------------------
# The following are for the PGS_GCT tool only.  The IDs are #defined in
# the PGS_GCT.h file.  These entries are essential for the State Plane
```

```
# Projection but can otherwise be deleted or commented out.
# ------------------------------------------------------------------------
10200|nad27sp|~/database/common/GCT||||1
10201|nad83sp|~/database/common/GCT||||1
# ------------------------------------------------------------------------
# The following are for the PGS_AA_DCW tool only.
# The IDs are #defined in the PGS_AA_DCW.h file.
# These entries may be deleted or commented out IF the AA tools are not
# being used.
# ------------------------------------------------------------------------
10990|eurnasia/|AA_DATA_INSTALL_DIR||||1
10991|noamer/|AA_DATA_INSTALL_DIR||||1
10992|soamafr/|AA_DATA_INSTALL_DIR||||1
10993|sasaus/|AA_DATA_INSTALL_DIR||||1
#
# ------------------------------------------------------------------------
# file for Constant & Unit Conversion (CUC) tools
# IMPORTANT NOTE: THIS FILE WILL BE SUPPLIED AFTER TK4 DELIVERY!
# ------------------------------------------------------------------------
10999|PGS_CUC_maths_parameters|~/database/common/CUC||||1
#
#
#------------------------------------------------------------------------
# Metadata Configuration File (MCF) is a template to be filled in by the
# Instrument teams.  MCFWrite.temp is a scratch file used to dump the MCF
# prior to writing to the hdf file. GetAttr.temp is similarly used to
# dump metadata from the hdf attributes and is used by PGS_MET_GetPCAttr.
# (DO NOT REMOVE THESE ENTRIES)
#------------------------------------------------------------------------
10250|MCF|||||1
```

```
10252|GetAttr.temp|||||1
10254|MCFWrite.temp|||||1
10260|XMLstylesheet.temp|||||1
#
#
# ---------------------------------------------------------------------
# Ephemeris and Attitude files logical IDs.
# Emphemeris files will be accessed via the logical ID 10501.
# Attitude files will be accessed via the logical ID 10502.
# Use file versions to allow for multiple physical ephemeris
# or attitude files.
# ---------------------------------------------------------------------
#
10501|INSERT_EPHEMERIS_FILES_HERE|||||1
10502|INSERT_ATTITUDE_FILES_HERE|||||1
10503|INSERT_EPHEMERIS_HDF_FILES_HERE|||||1
10504|INSERT_ATTITUDE_HDF_FILES_HERE|||||1
#
#---------------------------------------------------------------------
# Datasets for PGS_DEM tools.
# A dataset of a given resolution is accessed via a single logical ID,
# therefore all physical files comprising a data set must be accessed
# via the same logical ID.  Use file versions to allow for multiple
# physical files within a single data set.
# Data files of 30 arc-sec resolution will be accessed via the
# logical ID 10650.
# Data files of 3 arc-sec resolution will be accessed via the
# logical ID 10653.
# NOTE: The file names in each entry must also appear in the attribute
#       column of the entry (this is a requirement of the metadata tools).
```

```
#       The entries given below are "template" entries and should be
#       replaced with actual file name/location data before attempting
#       to use the DEM tools.
#-------------------------------------------------------------------------
#
10650|DEM30ARC_NAME.hdf|DEM_LOCATION|||DEM30ARC_NAME.hdf|1
10653|DEM3ARC_NAME.hdf|DEM_LOCATION|||DEM3ARC_NAME.hdf|1
#
?   PRODUCT OUTPUT FILES
##########################################################################
#
# This section is intended for standard product outputs, i.e., HDF-EOS
# files generated by this PGE.
#
# Each logical ID may have several file instances, as given by the
# version number in the last field.
#
##########################################################################
#
# Next line is the default location for PRODUCT OUTPUT FILES
!  ~/runtime
#
#-------------------------------------------------------------------------
# This file is created when PGS_MET_Write is used with an intention
# to write an ASCII representation of the MCF in memory. The user is
# allowed to change the name and path if required.
#
# NOTE: THIS IS OBSOLETE, THIS ENTRY IS ONLY HERE FOR BACKWARD
#       COMPATIBILITY WITH PREVIOUS VERSIONS OF THE TOOLKIT.
#       THE LOGICAL ID 10255 SHOULD BE MOVED DOWN TO THE RUNTIME
```

```
#       PARAMETERS SECTION OF THIS FILE AND GIVEN A VALUE OF:
#       <logical_id>:<version_number> WHERE THOSE VALUES REFLECT THE
#       ACTUAL VALUES FOR THE NON-HDF OUTPUT PRODUCT FOR WHICH THE
#       ASCII METADATA IS BEING WRITTEN.  e.g.:
#       10255|reference output product|100:2
#
#--------------------------------------------------------------------------
10255|asciidump|||||1
# -------------------------------------------------------------------------
#
?   SUPPORT INPUT FILES
##############################################################################
#
# This section is intended for minor input files, e.g., calibration
# files.
#
# Each logical ID may have several file instances, as given by the
# version number in the last field.
#
##############################################################################
#
# Next line is the default location for SUPPORT INPUT FILES
!  ~/runtime
#
#
# -------------------------------------------------------------------------
# This ID is #defined in PGS_AA_Tools.h
# This file contains the IDs for all support and format files (listed
# below).  This entry may be deleted or commented out if the AA tools are
# not being used.
```

```
# ---------------------------------------------------------------------
10900|indexFile|~/database/common/AA||||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 (above).
# These entries may be deleted or commented out if the AA tools are not
# being used.
# ---------------------------------------------------------------------
10901|mowe13aSupport|~/database/common/AA||||1
10902|owe13aSupport|~/database/common/AA||||1
10903|owe14Support|~/database/common/AA||||1
10904|etop05Support|~/database/common/AA||||1
10905|fnoc1Support|~/database/common/AA||||1
10906|fnoc2Support|~/database/common/AA||||1
10907|zobler1Support|~/database/common/AA||||1
10908|zobler2Support|~/database/common/AA||||1
10909|nmcRucSupport|~/database/common/AA||||1
10915|tbaseSupport|~/database/common/AA||||1
10916|tbase1Support|~/database/common/AA||||1
10917|tbase2Support|~/database/common/AA||||1
10918|tbase3Support|~/database/common/AA||||1
10919|tbase4Support|~/database/common/AA||||1
10740|usatile1Support|~/database/common/AA||||1
10741|usatile2Support|~/database/common/AA||||1
10742|usatile3Support|~/database/common/AA||||1
10743|usatile4Support|~/database/common/AA||||1
10744|usatile5Support|~/database/common/AA||||1
10745|usatile6Support|~/database/common/AA||||1
```

```
10746|usatile7Support|~/database/common/AA||||1
10747|usatile8Support|~/database/common/AA||||1
10748|usatile9Support|~/database/common/AA||||1
10749|usatile10Support|~/database/common/AA||||1
10750|usatile11Support|~/database/common/AA||||1
10751|usatile12Support|~/database/common/AA||||1
10948|geoidSupport|~/database/common/AA||||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 (10900, above).
# These entries may be deleted or commented out if the AA tools are not
# being used.
# ------------------------------------------------------------------------
10920|mowe13a.bfm|~/database/common/AA||||1
10921|owe13a.bfm|~/database/common/AA||||1
10922|owe14d.bfm|~/database/common/AA||||1
10923|owe14dr.bfm|~/database/common/AA||||1
10924|etop05.bfm|~/database/common/AA||||1
10925|fnocAzm.bfm|~/database/common/AA||||1
10926|fnocOcm.bfm|~/database/common/AA||||1
10927|fnocPt.bfm|~/database/common/AA||||1
10928|fnocRdg.bfm|~/database/common/AA||||1
10929|fnocSt.bfm|~/database/common/AA||||1
10930|fnocUrb.bfm|~/database/common/AA||||1
10931|fnocWat.bfm|~/database/common/AA||||1
10932|fnocMax.bfm|~/database/common/AA||||1
10933|fnocMin.bfm|~/database/common/AA||||1
10934|fnocMod.bfm|~/database/common/AA||||1
```

```
10935|srzArea.bfm|~/database/common/AA||||1
10936|srzCode.bfm|~/database/common/AA||||1
10937|srzPhas.bfm|~/database/common/AA||||1
10938|srzSlop.bfm|~/database/common/AA||||1
10939|srzSoil.bfm|~/database/common/AA||||1
10940|srzText.bfm|~/database/common/AA||||1
10941|nmcRucSigPotPres.bfm|~/database/common/AA||||1
10942|tbase.bfm|~/database/common/AA||||1
10943|tbase1.bfm|~/database/common/AA||||1
10944|tbase2.bfm|~/database/common/AA||||1
10945|tbase3.bfm|~/database/common/AA||||1
10946|tbase4.bfm|~/database/common/AA||||1
10700|usatile1.bfm|~/database/common/AA||||1
10701|usatile2.bfm|~/database/common/AA||||1
10702|usatile3.bfm|~/database/common/AA||||1
10703|usatile4.bfm|~/database/common/AA||||1
10704|usatile5.bfm|~/database/common/AA||||1
10705|usatile6.bfm|~/database/common/AA||||1
10706|usatile7.bfm|~/database/common/AA||||1
10707|usatile8.bfm|~/database/common/AA||||1
10708|usatile9.bfm|~/database/common/AA||||1
10709|usatile10.bfm|~/database/common/AA||||1
10710|usatile11.bfm|~/database/common/AA||||1
10711|usatile12.bfm|~/database/common/AA||||1
10947|geoid.bfm|~/database/common/AA||||1
#
#
# ------------------------------------------------------------------------
# leap seconds (TAI-UTC) file (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
```

```
10301|leapsec.dat|~/database/common/TD||||1
#
# ------------------------------------------------------------------------
# polar motion and UTC-UT1 file (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10401|utcpole.dat|~/database/common/CSC||||1
#
# ------------------------------------------------------------------------
# earth model tags file (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10402|earthfigure.dat|~/database/common/CSC||||1
#
# ------------------------------------------------------------------------
# JPL planetary ephemeris file (binary form) (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10601|de200.eos|~/database/$BRAND/CBP||||1
#
# ------------------------------------------------------------------------
# spacecraft tag definition file (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10801|sc_tags.dat|~/database/common/EPH||||1
#
# ------------------------------------------------------------------------
# units conversion definition file (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10302|udunits.dat|~/database/common/CUC||||1
#
# ------------------------------------------------------------------------
# Style Sheet for XML INVENTORY Metadata (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
```

```
10303|science.xsl|~/database/common/MET||||1
#
#
?   SUPPORT OUTPUT FILES
##############################################################################
#
# This section is intended for minor output files, e.g., log files.
#
# Each logical ID may have several file instances, as given by the
# version number in the last field.
#
##############################################################################
#
# Next line is default location for SUPPORT OUTPUT FILES
!  ~/runtime
#
#
# ------------------------------------------------------------------------
# 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. (DO NOT REMOVE THESE ENTRIES)
# ------------------------------------------------------------------------
10100|LogStatus|||||1
10101|LogReport|||||1
10102|LogUser|||||1
10103|TmpStatus|||||1
10104|TmpReport|||||1
10105|TmpUser|||||1
```

```
10110|MailFile|||||1
#
# ------------------------------------------------------------------------
# ASCII file which stores pointers to runtime SMF files in lieu of
# loading them to shared memory, which is a TK5 enhancement.
# (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10111|ShmMem|||||1
#
#
?   USER DEFINED RUNTIME PARAMETERS
##############################################################################
#
# This section is intended for parameters used as PGE input.
#
# Note: these parameters may NOT be changed dynamically.
#
##############################################################################
#
#
# ------------------------------------------------------------------------
# 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. (DO NOT REMOVE THESE ENTRIES)
# ------------------------------------------------------------------------
10109|TransmitFlag; 1=transmit,0=disable|0
10106|RemoteHost|sandcrab
10107|RemotePath|/usr/kwan/test/PC/data
```

```
10108|EmailAddresses|kwan@eos.hitc.com
#
# ------------------------------------------------------------------------
# The following runtime parameters define various logging options.
# Parameters described as lists should be space (i.e. ' ') separated.
# The logical IDs 10117, 10118, 10119 listed below are for OPTIONAL
# control of SMF logging.  Any of these logical IDs which is unused by a
# PGE may be safely commented out (e.g. if logging is not disabled for
# any status level, then the line beginning 10117 may be commented out).
# ------------------------------------------------------------------------
10114|Logging Control; 0=disable logging, 1=enable logging|1
10115|Trace Control; 0=no trace, 1=error trace, 2=full trace|0
10116|Process ID logging; 0=don't log PID, 1=log PID|0
10117|Disabled status level list (e.g. W S F)|
10118|Disabled seed list|
10119|Disabled status code list|
#
# ------------------------------------------------------------------------
# Toolkit version for which this PCF was intended.
# DO NOT REMOVE THIS VERSION ENTRY!
# ------------------------------------------------------------------------
10220|Toolkit version string|TK_VERSION_STRING
#
# ------------------------------------------------------------------------
# The following parameters define the ADEOS-II TMDF values (all values
# are assumed to be floating point types).  The ground reference time
# should be in TAI93 format (SI seconds since 12 AM UTC 1993-01-01).
# These formats are only prototypes and are subject to change when
# the ADEOS-II TMDF values are clearly defined.  PGEs that do not access
# ADEOS-II L0 data files do not require these parameters.  In this case
```

```
# they may be safely commented out, otherwise appropriate values should
# be supplied.
# ---------------------------------------------------------------------
10120|ADEOS-II s/c reference time|
10121|ADEOS-II ground reference time|
10122|ADEOS-II s/c clock period|
#
# ---------------------------------------------------------------------
# The following parameter defines the TRMM UTCF value (the value is
# assumed to be a floating point type).  PGEs that do not access TRMM
# data of any sort do not require this parameter.  In this case it may be
# safely commented out, otherwise an appropriate value should be
# supplied.
# ---------------------------------------------------------------------
10123|TRMM UTCF value|
#
# ---------------------------------------------------------------------
# The following parameter defines the Epoch date to be used for the
# interpretation (conversion) of NASA PB5C times (the Epoch date should
# be specified here in CCSDS ASCII format--A or B) (reserved for future
# use--this quantity is not referenced in TK 5.2).  This entry may be
# safely commented out or deleted.
# ---------------------------------------------------------------------
10124|NASA PB5C time Epoch date (ASCII UTC)|
#
# ---------------------------------------------------------------------
# The following parameter is a "mask" for the ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
```

```
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ------------------------------------------------------------------------
# The following parameter is a "mask" for the attitude data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the attitude data quality flag that
# should be considered fatal (i.e. the attitude data associated
# with the quality flag should be REJECTED/IGNORED).
# ------------------------------------------------------------------------
10508|attitude data quality flag mask|65536
#
# ------------------------------------------------------------------------
# ECS DPS trigger for PGE debug runs
#
# NOTICE TO PGE DEVELOPERS: PGEs which have a debug mode
# need to examine this parameter to evaluate activation rule
# (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10911|ECS DEBUG; 0=normal, 1=debug|0
#
# ------------------------------------------------------------------------
# The following runtime parameters defines generation of XML metadata during
# the production of .met file for the INVENTORY Metadata. If the flag is
# set to 0 only ASCII .met file will be created besides writing metadata into
# HDF file.If the flag is set to 1 then a .xml file will also be created
# in addition to ASCII .met file and metadata that is put into the HDF file.
# ------------------------------------------------------------------------
10256|XML METADATA GENERATION FLAG; 0=no, 1=yes|0
```

```
#
# ------------------------------------------------------------------------
# This entry defines the IP address of the processing host and is used
# by the Toolkit when generating unique Intermediate and Temporary file
# names.  The Toolkit no longer relies on the PGS_HOST_PATH environment
# variable to otain this information. (DO NOT REMOVE THIS ENTRY)
# ------------------------------------------------------------------------
10099|Local IP Address of 'ether'|155.157.31.87
#
?   INTERMEDIATE INPUT
##############################################################################
#
# This section is intended for intermediate input files, i.e., files
# which are output by an earlier PGE but which are not standard
# products.
#
# Each logical ID may have only one file instance.
# Last field on the line is ignored.
#
##############################################################################
#
# Next line is default location for INTERMEDIATE INPUT FILES
!  ~/runtime
#
#
?   INTERMEDIATE OUTPUT
##############################################################################
#
# This section is intended for intermediate output files, i.e., files
# which are to be input to later PGEs, but which are not standard
```

```
# products.
#
# Each logical ID may have only one file instance.
# Last field on the line is ignored.
#
##############################################################################
#
# Next line is default location for INTERMEDIATE OUTPUT FILES
!  ~/runtime
#
#
?   TEMPORARY I/O
##############################################################################
#
# This section is intended for temporary files, i.e., files
# which are generated during a PGE run and deleted at PGE termination.
#
# Entries in this section are generated internally by the Toolkit.
# DO NOT MAKE MANUAL ENTRIES IN THIS SECTION.
#
##############################################################################
#
# Next line is default location for TEMPORARY FILES
!  ~/runtime
#
#
?   END
```

C.2 Validating Process Control Files

C.2.1 DESCRIPTION

The Process Control Information File Check Program is a program that checks the file containing the Process Control Status Information. This program is an aid to determine if the input file necessary for the Process Control Tools is in the proper format and contains the minimum amount of information for a valid run. The program is run by entering the program name followed by the file name to be checked. For example, "pccheck.sh -i userpcf.dat" will run the check program and check the file userpcf.dat located in the current directory. The -i flag needs to be followed by the name of the input file. Upon checking the file, a list of errors and warnings will be displayed to the user. Each error or warning will have a brief description, the line number, and the line itself. When the checking process has completed, a message appears stating that the check process is finished and the number of warnings and errors found are displayed. With this program, errors are defined as something in the file that, during execution of the Process Control Tools, the return will not be PGS_S_SUCCESS. A warning is defined as something that, although the Process Control Tools will return a PGS_S_SUCCESS, a problem could arise later. An example of this is the file name "file one.dat" is stored in the Process Control Information file. Upon execution, the Process Control Tools will return the name of this file and PGS_S_SUCCESS as the function type return value. When the program tries to open this file however, a file access error will occur.

C.2.2 INPUT

- Program name, -i flag, and file to be checked. An example of this would be:

```
pccheck.sh -i userpcf.dat
```

This will initiate the check program and check the file userpcf.dat in the current directory.

- Program name, -i flag, file to be checked, -o flag, and an output file name.

```
pccheck.sh -i userpcf.dat -o userpcf.out
```

This will initiate the check program and check the file userpcf.dat in the current directory and create an output file "outpct.fil" that will be an exact copy of userpcf.dat except the output file will contain line numbers.

- Program name, -h flag.

```
pccheck.sh -h
```

This will display a usage help message.

- Program name, -i flag, file to be checked, -c flag, and a template file name.

```
pccheck.sh -i userpcf.dat -c $PGSHOME/runtime/PCF.v3
```

This will list all errors and warnings in the file userpcf.dat and perform a comparison. The -c flag will initiate a comparison with a template file and determine if any of the Product ID's reserved

by the PGS Toolkit (range 10,000 .. 10,999) differ in userpcf.dat and $PGSHOME/runtime/PCF.v3. This will only list the differences and will not perform any corrections.

- Program name, -i flag, file to be checked, -c flag, and a template file name, -s flag, to suppress output.

```
pccheck.sh -i userpcf.dat -c $PGSHOME/runtime/PCF.v3 -s
```

The -s flag will suppress all output except that output received when using the -c flag. The -s flag is designed to be used only when the -c flag is used.

C.2.3 OUTPUT

List of errors and warnings are followed by a summary of the number of errors and warnings. See the EXAMPLES section for detailed listings of program output. Using the -o flag will also allow the user to output a file that is an exact copy of the input file with line numbers in the file. This output option is provided as a convenience to the user; the output file is not intended to be used as the input Process Control Information File. Using the -c flag followed by a template file will allow the user to determine what reserved Logical Identifiers have been edited from the template file.

C.2.4 ERRORS

The following is a list of possible errors followed by a brief description.

- "Unable to open input file: <file name>"—unable to open input file name passed in as a command line argument
- "Incorrect number of command line arguments"—the number of command line arguments did not match the number expected
- "Unexpectedly reached EOF"—the end of file was encountered before the correct number of dividers (?) were reached
- "Invalid number of system configuration parameters"—the number of system configuration parameters encountered did not match the number expected
- "Invalid index value in user defined configuration parameters"—an invalid index value was found
- "Problem with user defined configuration parameter"—user defined configuration parameter contains a problem (i.e., incorrect number of delimiters (|), or a value of all blanks)
- "Configuration value length too long"—user defined configuration value exceeds PGSd_PC_VALUE_LENGTH_MAX characters
- "Invalid index value involving file information"—an invalid index value was found in one of the sections that contains file information

- "Invalid number of delimiters involving file information"—line containing file information contains incorrect number of delimiters (|)
- "No validity flag present in input file information"—validity flag is mandatory for input file information
- "File name length too long"—file name exceeds PGSd_PC_FILE_NAME_MAX characters
- "Path length too long"—path exceeds PGSd_PC_PATH_LENGTH_MAX characters
- "problem with version number in Standard input file information"—missing or unexpected sequence number
- "Default file location marker contains no data."
- "Default file location length too long."
- "Default file location not found."
- "Universal Reference length too long." - universal reference identifier exceeds PGSd_PC_UREF_LENGTH_MAX characters
- “File name does not exist.” - File name data field is empty.

C.2.5 WARNINGS

The following is a list of all possible warnings followed by a brief description.

- "Warning—Possible problem with system configuration value"—configuration parameter contains all blank characters
- "Warning—Repeat index number in user defined configuration parameters"—index value used twice in user defined configuration parameters
- "Warning—extra delimiters in user defined configuration parameters"—remaining delimiters will be returned as part of the value in user defined configuration parameters.
- "Warning—Repeat index number in file information"—index value illegally used multiple times in file information
- "Warning—possible problem in path or file name"—path or file name contains blank characters
- "Warning—information beyond final divider will be ignored"—anything after the last counted divider (?) will be ignored
- "Warning—possible problem in default file location."
- "Warning—Default file location not after divider."

C.2.6 EXAMPLES

Three examples are provided below. Each example contains the input file used, the command entered and the corresponding output. The first example contains no errors or warnings. The second example contains several warnings and errors. The third example is an example of using the -c flag.

C.2.6.1 EXAMPLE 1

INPUT FILE: userpcf.dat

```
#
#   Process Control File
#
#
?   SYSTEM RUNTIME PARAMETERS
# ---------------------------------------------------------------------
# Production Run ID - unique production instance identifier
# ---------------------------------------------------------------------
1
# ---------------------------------------------------------------------
# Software ID - unique software configuration identifier
# ---------------------------------------------------------------------
1
#
?   PRODUCT INPUT FILES
# [ Default file location indicated by '!' ]
! ~/runtime
#
1000|temp.dat|/usr/atm/data||Optional Universal Reference|temp.att|1
1001|humid.dat|/usr/atm/data||Humidity Data|humid.att|1
600|wind_1.dat||||wind_1.att|2
600|wind_2.dat||||wind_2.att|1
# ---------------------------------------------------------------------
# polar motion and UTC-UT1 file
# ---------------------------------------------------------------------
10401|utcpole.dat|~/lib/database/CSC||||1
# ---------------------------------------------------------------------
# earth model tags file
# ---------------------------------------------------------------------
10402|earthfigure.dat|~/lib/database/CSC||||1
# ---------------------------------------------------------------------
# JPL planetary ephemeris file (binary form)
# ---------------------------------------------------------------------
10601|de200.eos|/usr/lib/database/CBP||||1
10964|fnocmin.imgswitched|||||1
```

```
10965|fnocmod.imgswitched|||||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
#
# ------------------------------------------------------------------------
# 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
#
#
?   PRODUCT OUTPUT FILES
# [ Default file location indicated by '!' ]
! ~/runtime
#
1002|temp_lev3.hdf|||||1
1003|humid_lev3.hdf|||||1
601|wind_lev3.hdf|||||1
#
#
?   SUPPORT INPUT FILES
# [ Default file location indicated by '!' ]
! ~/runtime
#
31|Wind_insitu.dat|/usr/wind/data||||1
#
#
# ------------------------------------------------------------------------
# This ID is #defined in PGS_AA_Tools.h
# This file contains the IDs for all support and format files shown
# above
# ------------------------------------------------------------------------
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
#
# ------------------------------------------------------------------------
# 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
#
#
?   SUPPORT OUTPUT FILES
# [ Default file location indicated by '!' ]
! ~/runtime
#
#
51|Wind_qlook.dat|/usr/wind/data||||1
#
```

```
# ------------------------------------------------------------------------
# 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
#
#
?   USER DEFINED RUNTIME PARAMETERS
3000|Humidity Instrument Calibration|0.34423772
3001|Temperature Instrument Calibration|1.87864
3002|Wind Instrument Calibration|0.992
3003|Atmospheric Algorithm|NIGHT
3004|Status Report Title|INSTRUMENT STATUS REPORT FOR LEVEL 2
#
#
# ------------------------------------------------------------------------
# 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|0
10106|RemoteHost|anyhost
10107|RemotePath|/usr/anyuser/anypath/data
10108|EmailAddresses|anyuser@anysystem.anyaddress.gov
#
#
?   INTERMEDIATE INPUT
# [ Default file location indicated by '!' ]
! ~/runtime
#
#
?   INTERMEDIATE OUTPUT
# [ Default file location indicated by '!' ]
! ~/runtime
```

```
#
#
?   TEMPORARY IO
# [ Default file location indicated by '!' ]
! ~/runtime
#
#
?   END
```

UNIX COMMAND LINE:

```
pccheck.sh -i userpcf.dat

Check of userpcf.dat completed
Errors found:  0
Warnings found:  0
```

C.2.6.2 EXAMPLE 2

INPUT FILE: userpcf.dat

```
#
#   Process Control File
#
#
?   SYSTEM RUNTIME PARAMETERS
# ---------------------------------------------------------------------
# Production Run ID - unique production instance identifier
# ---------------------------------------------------------------------
1
#****ONLY ONE SYSTEM CONFIGURATION PARAMETER****
#
?   PRODUCT INPUT FILES
# [ Default file location  marked by '!' ]
! ~/runtime
#
1000|temp.dat|/usr/atm/data|||temp.att|
#                                     ^ No version number****
1)01|humid.dat|/usr/atm/data|||humid.att|1
#^Illegal character in index number****
600|wind_1.dat||||wind_1.att|2
600|wind_2.dat|||wind_2.att|1
#  Line only contains five delimiters****
#
# ---------------------------------------------------------------------
# polar motion and UTC-UT1 file
# ---------------------------------------------------------------------
```

```
10401|utcpole.dat|~/lib/database/CSC||||1
# ------------------------------------------------------------------------
# earth model tags file
# ------------------------------------------------------------------------
10402|earthfigure.dat|~/lib/database/CSC||||1
# ------------------------------------------------------------------------
# JPL planetary ephemeris file (binary form)
# ------------------------------------------------------------------------
10601|de200.eos|/usr/lib/database/CBP||||1
10964|fnocmin.imgswitched|||||1
10965|fnocmod.imgswitched|||||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
#
# ------------------------------------------------------------------------
# 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
#
#
?   PRODUCT OUTPUT FILES
#
# ^^^^ No default file location listed before first file name****
1002|temp_lev3.hdf|||||1
1003|humid_lev3.hdf|||||1
601|wind_lev3.hdf|||||1
#
#
?   SUPPORT INPUT FILES
# [ Default file location  marked by '!' ]
! ~/runtime
#
31|Wind_insitu .dat|/usr/wind/data||||1
#             ^ Blank character in file name****
#
#
# ------------------------------------------------------------------------
```

```
# This ID is #defined in PGS_AA_Tools.h
# This file contains the IDs for all support and format files shown
# above
# ------------------------------------------------------------------------
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
#
# ------------------------------------------------------------------------
# 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
#
#
?   SUPPORT OUTPUT FILES
# [ Default file location  marked by '!' ]
! ~/runtime
#
#
#
51|Wind_qlook.dat|/usr/wind/data||||1
#
# ------------------------------------------------------------------------
# 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
#
#
?   USER DEFINED RUNTIME PARAMETERS
3000|Humidity Instrument Calibration|0.34423772
3001|
#    ^ Incomplete line****
3002|Wind Instrument Calibration|0.992|
#                                    ^ Extra delimiter****
3003|Atmospheric Algorithm|NIGHT
3001|Status Report Title|INSTRUMENT STATUS REPORT FOR LEVEL 2
#    Index number used six lines above****
#
#
# ------------------------------------------------------------------------
# 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|0
10106|RemoteHost|anyhost
10107|RemotePath|/usr/anyuser/anypath/data
10108|EmailAddresses|anyuser@anysystem.anyaddress.gov
#
#
?   INTERMEDIATE INPUT
# [ Default file location  marked by '!' ]
! ~/runtime
#
#
#
?   INTERMEDIATE OUTPUT
# [ Default file location  marked by '!' ]
! ~/runtime
#
#
?   TEMPORARY IO
# [ Default file location  marked by '!' ]
! ~/runtime
#
#
?   END
#       We just passed the last divider****
```

UNIX COMMAND LINE:

```
pccheck.sh -i userpcf.dat -o userpcf.out
Error - Invalid number of system configuration parameters.
Found:  1
Expected:  2

Error - problem with version number in Standard input or output file
information.
Line number:  16
Line:  1000|temp.dat|/usr/atm/data|||temp.att|

Error - Invalid identifier number involving file information.
Line number:  18
Line:  1)01|humid.dat|/usr/atm/data|||humid.att|1

Error - Invalid number of delimiters involving file information.
Line number:  21
Line:  600|wind_2.dat|||wind_2.att|1
```

```
Error - Default file location not found.
Line number:  58
Line:  1002|temp_lev3.hdf|||||1

Warning - possible problem in path or file name.
Line number:  67
Line:  31|Wind_insitu .dat|/usr/wind/data||||1

Error - Problem with user defined configuration parameter.
Line number:  146
Line:  3001|

Warning - extra delimiters in user defined configuration parameters.
Line number:  148
Line:  3002|Wind Instrument Calibration|0.992|

Warning - Repeat index number in user defined configuration parameters.
Line number:  151
Line:  3001|Status Report Title|INSTRUMENT STATUS REPORT FOR LEVEL 2

Warning - information beyond final divider will be ignored.
line number:  185
Number of dividers read:  10
Number of dividers expected:  10


Check of usrpcf.dat completed
Errors found:  6
Warnings found:  4
```

OUTPUT FILE: userpcf.out

```
  1:#
  2:#   Process Control File
  3:#
  4:#
  5:?   SYSTEM RUNTIME PARAMETERS
  6:# ---------------------------------------------------------------------
  7:# Production Run ID - unique production instance identifier
  8:# ---------------------------------------------------------------------
  9:1
 10:#****ONLY ONE SYSTEM CONFIGURATION PARAMETER****
 11:#
 12:?   PRODUCT INPUT FILES
 13:# [ Default file location  marked by '!' ]
```

```
14:! ~/runtime
15:#
16:1000|temp.dat|/usr/atm/data|||temp.att|
17:#                                      ^ No version number****
18:1)01|humid.dat|/usr/atm/data|||humid.att|1
19:#^Illegal character in index number****
20:600|wind_1.dat||||wind_1.att|2
21:600|wind_2.dat|||wind_2.att|1
22:#  Line only contains five delimiters****
23:#
24:# ---------------------------------------------------------------------
25:# polar motion and UTC-UT1 file
26:# ---------------------------------------------------------------------
27:10401|utcpole.dat|~/lib/database/CSC||||1
28:# ---------------------------------------------------------------------
29:# earth model tags file
30:# ---------------------------------------------------------------------
31:10402|earthfigure.dat|~/lib/database/CSC||||1
32:# ---------------------------------------------------------------------
33:# JPL planetary ephemeris file (binary form)
34:# ---------------------------------------------------------------------
35:10601|de200.eos|/usr/lib/database/CBP||||1
36:10964|fnocmin.imgswitched|||||1
37:10965|fnocmod.imgswitched|||||1
38:10966|srzarea.img|||||1
39:10967|srzcode.img|||||1
40:10968|srzphas.img|||||1
41:10969|srzslop.img|||||1
42:10970|srzsoil.img|||||1
43:10971|srztext.img|||||1
44:#
45:# ---------------------------------------------------------------------
46:# The following are for the PGS_AA_dcw tool only.
47:# The IDs are #defined in the PGS_AA_dcw.h file
48:# ---------------------------------------------------------------------
49:10990|eurnasia/|||||1
50:10991|noamer/|||||1
51:10992|soamafr/|||||1
52:10993|sasaus/|||||1
53:#
54:#
55:?   PRODUCT OUTPUT FILES
56:#
57:# ^^^^ No default file location listed before first file name****
58:1002|temp_lev3.hdf|||||1
```

```
 59:1003|humid_lev3.hdf|||||1
 60:601|wind_lev3.hdf|||||1
 61:#
 62:#
 63:?   SUPPORT INPUT FILES
 64:# [ Default file location  marked by '!' ]
 65:! ~/runtime
 66:#
 67:31|Wind_insitu .dat|/usr/wind/data||||1
 68:#             ^ Blank character in file name****
 69:#
 70:#
 71:# ---------------------------------------------------------------------
 72:# This ID is #defined in PGS_AA_Tools.h
 73:# This file contains the IDs for all support and format files shown
 74:# above
 75:# ---------------------------------------------------------------------
 76:10900|indexFile|~/runtime||||1
 77:#
 78:# ---------------------------------------------------------------------
 79:# These are support files for the data set files - to be created by user
 80:# (not necessarily a one-to-one relationship)
 81:# The IDs must correspond to the logical IDs in the index file
 82:# ---------------------------------------------------------------------
 83:10901|mowe13aSupport|~/runtime||||1
 84:10902|owe13aSupport|~/runtime||||1
 85:10903|owe14Support|~/runtime||||1
 86:10904|etop05Support|~/runtime||||1
 87:10905|fnoc1Support|~/runtime||||1
 88:10906|fnoc2Support|~/runtime||||1
 89:10907|zobler1Support|~/runtime||||1
 90:10908|zobler2Support|~/runtime||||1
 91:#
 92:# ---------------------------------------------------------------------
 93:# The following are format files for each data set file
 94:# (not necessarily a one-to-one relationship)
 95:# The IDs must correspond to the logical IDs in the index file
 96:# ---------------------------------------------------------------------
 97:10920|mowe13a.bfm|~/runtime||||1
 98:10921|owe13a.bfm|~/runtime||||1
 99:10922|owe14d.bfm|~/runtime||||1
100:10923|owe14dr.bfm|~/runtime||||1
101:10924|etop05.bfm|~/runtime||||1
102:10925|fnocAzm.bfm|~/runtime||||1
103:10926|fnocOcm.bfm|~/runtime||||1
```

```
104:10927|fnocPt.bfm|~/runtime||||1
105:10928|fnocRdg.bfm|~/runtime||||1
106:10929|fnocSt.bfm|~/runtime||||1
107:10930|fnocUrb.bfm|~/runtime||||1
108:10931|fnocWat.bfm|~/runtime||||1
109:10932|fnocMax.bfm|~/runtime||||1
110:10933|fnocMin.bfm|~/runtime||||1
111:10934|fnocMod.bfm|~/runtime||||1
112:10935|srzArea.bfm|~/runtime||||1
113:10936|srzCode.bfm|~/runtime||||1
114:10937|srzPhas.bfm|~/runtime||||1
115:10938|srzSlop.bfm|~/runtime||||1
116:10939|srzSoil.bfm|~/runtime||||1
117:10940|srzText.bfm|~/runtime||||1
118:#
119:#
120:?   SUPPORT OUTPUT FILES
121:# [ Default file location  marked by '!' ]
122:! ~/runtime
123:#
124:#
125:#
126:51|Wind_qlook.dat|/usr/wind/data||||1
127:#
128:# ------------------------------------------------------------------------
129:# These files support the SMF log functionality. Each run will cause
130:# status information to be written to 1 or more of the Log files. To
131:# simulate DAAC operations, remove the 3 Logfiles between test runs.
132:# Remember: all executables within a PGE will contribute status data to
133:# the same batch of log files.
134:# ------------------------------------------------------------------------
135:10100|LogStatus|~/runtime||||1
136:10101|LogReport|~/runtime||||1
137:10102|LogUser|~/runtime||||1
138:10103|TmpStatus|~/runtime||||1
139:10104|TmpReport|~/runtime||||1
140:10105|TmpUser|~/runtime||||1
141:10110|MailFile|~/runtime||||1
142:#
143:#
144:?   USER DEFINED RUNTIME PARAMETERS
145:3000|Humidity Instrument Calibration|0.34423772
146:3001|
147:#    ^ Incomplete line****
148:3002|Wind Instrument Calibration|0.992|
```

```
149:#                                     ^ Extra delimiter****
150:3003|Atmospheric Algorithm|NIGHT
151:3001|Status Report Title|INSTRUMENT STATUS REPORT FOR LEVEL 2
152:#    Index number used six lines above****
153:#
154:#
155:# ------------------------------------------------------------------------
156:# These parameters are required to support the PGS_SMF_Send...() tools.
157:# If the first parameter (TransmitFlag) is disabled, then none of the
158:# other parameters need to be set. By default, this functionality has
159:# been disabled. To enable, set TransmitFlag to 1 and supply the other 3
160:# parameters with local information.
161:# ------------------------------------------------------------------------
162:10109|TransmitFlag; 1=transmit,0=disable|0
163:10106|RemoteHost|anyhost
164:10107|RemotePath|/usr/anyuser/anypath/data
165:10108|EmailAddresses|anyuser@anysystem.anyaddress.gov
166:#
167:#
168:?   INTERMEDIATE INPUT
169:# [ Default file location  marked by '!' ]
170:! ~/runtime
171:#
172:#
173:#
174:?   INTERMEDIATE OUTPUT
175:# [ Default file location  marked by '!' ]
176:! ~/runtime
177:#
178:#
179:?   TEMPORARY IO
180:# [ Default file location  marked by '!' ]
181:! ~/runtime
182:#
183:#
184:?   END
185:#       We just passed the last divider****
```

C.2.6.3 EXAMPLE 3

INPUT FILE: userpcf.dat

```
#
#   Process Control File
#
```

```
#
?   SYSTEM RUNTIME PARAMETERS
# ---------------------------------------------------------------------
# Production Run ID - unique production instance identifier
# ---------------------------------------------------------------------
1
# ---------------------------------------------------------------------
# Software ID - unique software configuration identifier
# ---------------------------------------------------------------------
1
#
?   PRODUCT INPUT FILES
# [ Default file location marked by '!' ]
! ~/runtime
# ---------------------------------------------------------------------
# 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.
# ---------------------------------------------------------------------
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
10975|geoid.dat|||||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
#
1000|temp.dat|/usr/atm/data|||temp.att|1
1001|humid.dat|/usr/atm/data|||humid.att|1
600|wind_1.dat||||wind_1.att|2
600|wind_2.dat||||wind_2.att|1
# ------------------------------------------------------------------------
# polar motion and UTC-UT1 file
# ------------------------------------------------------------------------
10401|utcpole.dat|~/lib/database/CSC||||1
# ------------------------------------------------------------------------
# earth model tags file
# ------------------------------------------------------------------------
10402|earthfigure.dat|~/lib/database/CSC||||1
# ------------------------------------------------------------------------
# JPL planetary ephemeris file (binary form)
# ------------------------------------------------------------------------
```

```
10601|de200.eos|/usr/lib/database/CBP||||1
#
#
?   PRODUCT OUTPUT FILES
# [ Default file location marked by '!' ]
! ~/runtime
#
1002|temp_lev3.hdf|||||1
1003|humid_lev3.hdf|||||1
601|wind_lev3.hdf|||||1
#
#
?   SUPPORT INPUT FILES
# [ Default file location marked by '!' ]
! ~/runtime
#
31|Wind_insitu.dat|/usr/wind/data||||1
#
#
# ------------------------------------------------------------------------
# This ID is #defined in PGS_AA_Tools.h
# This file contains the IDs for all support and format files shown
# above
# ------------------------------------------------------------------------
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
10948|geoidSupport|~/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
#
#
?   SUPPORT OUTPUT FILES
# [ Default file location marked by '!' ]
! ~/runtime
```

```
#
#
51|Wind_qlook.dat|/usr/wind/data||||1
#
# ------------------------------------------------------------------------
# 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
#
#
?   USER DEFINED RUNTIME PARAMETERS
3000|Humidity Instrument Calibration|0.34423772
3001|Temperature Instrument Calibration|1.87864
3002|Wind Instrument Calibration|0.992
3003|Atmospheric Algorithm|NIGHT
3004|Status Report Title|INSTRUMENT STATUS REPORT FOR LEVEL 2
#
#
# ------------------------------------------------------------------------
# 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|0
10106|RemoteHost|anyhost
10107|RemotePath|/usr/anyuser/anypath/data
10108|EmailAddresses|anyuser@anysystem.anyaddress.gov
#
#
?   INTERMEDIATE INPUT
# [ Default file location marked by '!' ]
! ~/runtime
#
```

```
#
?   INTERMEDIATE OUTPUT
# [ Default file location marked by '!' ]
! ~/runtime
#
#
?   TEMPORARY IO
# [ Default file location marked by '!' ]
! ~/runtime
#
#
?   END
```

COMPARISON FILE: PCF.testmaster

```
#
# filename:
#     PCF.testmaster
#
# description:
#     Process Control File (PCF)
#
# notes:
#
#     This file supports the IR-1 version of the toolkit.
#
#     Please treat this file as a master template and make copies of it
#     for your own testing. Note that the Toolkit installation script sets
#     PGS_PC_INFO_FILE to point to this master file by default. Remember
#     to reset the environment variable PGS_PC_INFO_FILE to point to the
#     instance of your PCF.
#
# ------------------------------------------------------------------------
?   SYSTEM RUNTIME PARAMETERS
# ------------------------------------------------------------------------
# Production Run ID - unique production instance identifier
# ------------------------------------------------------------------------
1
# ------------------------------------------------------------------------
# Software ID - unique software configuration identifier
# ------------------------------------------------------------------------
1
#
?   PRODUCT INPUT FILES
```

```
# Next non-comment line is the default location for PRODUCT INPUT FILES
# WARNING! DO NOT MODIFY THIS LINE unless you have relocated these
# data set files to the location specified by the new setting.
!  ~/runtime
#
# ------------------------------------------------------------------------
# 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.
# ------------------------------------------------------------------------
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
10975|geoid.dat|||||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
#
#
?   PRODUCT OUTPUT FILES
# Next line is the default location for PRODUCT OUTPUT FILES
!  ~/runtime
#
#
?   SUPPORT INPUT FILES
# Next line is the default location for SUPPORT INPUT FILES
!  ~/runtime
#
#
# ------------------------------------------------------------------------
# This ID is #defined in PGS_AA_Tools.h
# This file contains the IDs for all support and format files shown above
# ------------------------------------------------------------------------
10900|indexFile|~/runtime||||1
#
# ------------------------------------------------------------------------
# These are support files for the data set files - to be created by user
# (not necessary
# 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
10948|geoidSupport|~/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
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
10947|geoidbfm|~/runtime||||1
#
#
# ------------------------------------------------------------------------
# leap seconds (TAI-UTC) file
# ------------------------------------------------------------------------
10301|leapsec.dat|~/database/sun5/TD||||1
#
# ------------------------------------------------------------------------
# polar motion and UTC-UT1 file
# ------------------------------------------------------------------------
10401|utcpole.dat|~/database/sun5/CSC||||1
#
# ------------------------------------------------------------------------
# earth model tags file
# ------------------------------------------------------------------------
10402|earthfigure.dat|~/database/sun5/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|.|~/database/sun5/EPH||||1
#
# ------------------------------------------------------------------------
# JPL planetary ephemeris file (binary form)
# ------------------------------------------------------------------------
10601|de200.eos|~/database/sun5/CBP||||1
#
```

```
#
?   SUPPORT OUTPUT FILES
# Next line is default location for SUPPORT OUTPUT FILES
!  ~/runtime
#
#
# ------------------------------------------------------------------------
# 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|||||1
10101|LogReport|||||1
10102|LogUser|||||1
10103|TmpStatus|||||1
10104|TmpReport|||||1
10105|TmpUser|||||1
10110|MailFile|||||1
#
# ------------------------------------------------------------------------
# This parameter controls the Event Logger connection from the Toolkit.
# ------------------------------------------------------------------------
10113|eventLogger.log|||||1
#
# ------------------------------------------------------------------------
# ASCII file which stores pointers to runtime SMF files in lieu of
# loading them to shared memory, which is a TK5 enhancement.
# ------------------------------------------------------------------------
10111|ShmMem|||||1
#
#
?   USER DEFINED RUNTIME PARAMETERS
#
#
# ------------------------------------------------------------------------
# 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|0
10106|RemoteHost|sandcrab
```

```
10107|RemotePath|/usr/kwan/test/PC/data
10108|EmailAddresses|kwan@eos.hitc.com
#
# ------------------------------------------------------------------------
# This parameter controls the Event Logger connection from the Toolkit.
# ------------------------------------------------------------------------
10112|Event Logging Flag; 1=connect,0=disconnect|1
#
# ------------------------------------------------------------------------
# This entry defines the IP address of the processing host and is used
# by the Toolkit when generating unique Intermediate and Temporary file
# names.  The Toolkit no longer relies on the PGS_HOST_PATH environment
# variable to obtain this information.
# ------------------------------------------------------------------------
10099|Local IP Address of 'ether'|155.157.31.87
#
?   INTERMEDIATE INPUT
# Next line is default location for INTERMEDIATE INPUT FILES
!  ~/runtime
#
#
?   INTERMEDIATE OUTPUT
# Next line is default location for INTERMEDIATE OUTPUT FILES
!  ~/runtime
#
#
?   TEMPORARY I/O
# Next line is default location for TEMPORARY FILES
!  ~/runtime
#
#
?   END
```

UNIX COMMAND LINE:

```
pccheck.sh -i userpcf.dat -c PCF.testmaster -s
The following lines were listed in the template file:  PCF.testmaster
and have been altered or deleted from the input file.

> 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
> 10947|geoidbfm|~/runtime||||1
> 10301|leapsec.dat|~/database/sun5/TD||||1
> 10401|utcpole.dat|~/database/sun5/CSC||||1
> 10402|earthfigure.dat|~/database/sun5/CSC||||1
> 10501|.|~/database/sun5/EPH||||1
> 10601|de200.eos|~/database/sun5/CBP||||1
> 10100|LogStatus|||||1
> 10101|LogReport|||||1
> 10102|LogUser|||||1
> 10103|TmpStatus|||||1
> 10104|TmpReport|||||1
> 10105|TmpUser|||||1
> 10110|MailFile|||||1
> 10113|eventLogger.log|||||1
> 10111|ShmMem|||||1
> 10106|RemoteHost|sandcrab
> 10107|RemotePath|/usr/kwan/test/PC/data
> 10108|EmailAddresses|kwan@eos.hitc.com
> 10112|Event Logging Flag; 1=connect,0=disconnect|1
> 10099|Local IP Address of 'ether'|155.157.31.87

These are the lines in the input file:  usrpcf.dat
that differ from the template file.
< 10401|utcpole.dat|~/lib/database/CSC||||1
< 10402|earthfigure.dat|~/lib/database/CSC||||1
< 10601|de200.eos|/usr/lib/database/CBP||||1
< 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
< 10106|RemoteHost|anyhost
< 10107|RemotePath|/usr/anyuser/anypath/data
< 10108|EmailAddresses|anyuser@anysystem.anyaddress.gov
```

C.2.7 BENEFITS

Due to the fact that the Process Control Information file must currently be entered by hand, errors can easily be introduced. Many errors are not obvious and may not be detected by the Process Control Tools. By adopting the practice of using this utility to check your PCF after each modification, the number of runtime errors can be greatly reduced.

Appendix D. Ancillary Data Access Tools

This appendix deals with the use of the ancillary data access tools:

PGS_AA_dcw

PGS_AA_dem

PGS_AA_2DRead

PGS_AA_2Dgeo

PGS_AA_3DRead

PGS_AA_3Dgeo

PGS_AA_PeVA

The first section below describes how the tools are conceived. Each tool is then described in terms of

- the data set(s) to which it is designed to give access including its accuracy and precision
- an outline of the means by which the tool achieves access and any options available through the calling sequence.
- how the user can call the tool to optimize resource efficiency
- upgrade possibilities

The DCW tool is described in the second section; while the DEM, 2 and 3 D tools, being closely allied in functional terms are described together in the third section. The fourth section describes the Parameter = Value tool that is a support tool for the other tools but can also be used directly by science users in algorithms.

This information is additional to that in the main User Guide pages and calling sequence details are not repeated here.

D.1 Introduction

The ancillary data tools are optional for use in science algorithms. There is a wide range of ancillary data sets and these tools have been designed to provide useful access functionality only for those data sets for which generic functionality can be provided centrally.

Users could utilize language standard input/output functions or the HDF tools to access the ancillary data. However, a suite of higher level tools is required for four reasons:

a. to enable data from locations specified by the user to be returned to the user thus avoiding having to know the internal structure of the file.

b. to shield the user from having to know details of parameter source or source format or to track changes in either, although sources changes will be agreed with the user.

c. to provide for certain additional manipulations of extracted data.

For this final point (c), only those data sets that have been specifically identified as requiring particular manipulations will be serviced; i.e., the ancillary tools do not intend to provide a general manipulation service for all types of data. However, the tools which 'extract from location' (a) will be sufficiently generic to allow additional data sets of a similar type to be used.

Access to the information will be in response to an algorithm request in the form of pointers to parameters and locations in a data file. These pointers take the form of a latitude and longitude or a similar two dimensional or three dimensional pointer.

It has been assumed that users require to access single or multiple point locations for one or more parameters and that these values will be returned in arrays to the user. This is in sharp contrast to the other major use of various ancillary data that are used for display purposes on screens. It is further assumed that the user requires multiple extractions made in user defined loops; very often driven by the systematic examination of time ordered source packets along or orthogonal to the sub–satellite track.

D.2 PGS_AA_dcw

D.2.1 Data Sets Accessed

PGS_AA_dcw is an ancillary data tool to be used to access the Digital Chart of the World database (DCW). The tool can only be used for accessing DCW.

A subset of the DCW database subset is delivered with the tool. For descriptions of data sets and file structure see **Digital Chart of the World—Final DCW Product Specification MIL–D–89009 December 7, 1991**.

DCW is a general purpose digital global database designed for Geographical Information Systems (GIS) applications. It utilizes a vector based, thematically layered data set available on four CD–ROM's at a comprehensive scale of 1:1,000,000. It consists of geographic, attribute, and textual data, stored in Vector Product Format or VPF. VPF is described in **Vector Product Format (MIL–STD–600006)**.

The data provided with the tool is exactly as found in the product, therefore any errors are a result of the database and not the tool. The DCW content is based primarily on the feature content of the 1:1,000,000–scale DMA Operational Navigational Chart (ONC) series. The 270 ONC sheets are supplemented with six 1:2,000,000–scale Jet Navigation Charts (JNC's) in the Antarctic region where ONC coverage is not available.

The absolute horizontal accuracy of the DCW for all features derived from ONC's is < 2040 meters (<6700 feet) rounded to the nearest 5 meters at 90% Circular Error (CE), World Geodetic System (PGSD_WGS84). The absolute horizontal accuracy for all features derived from JNC's is <4270 meters (<14000 feet) at 90% CE.

DCW is provided normally on four CDROM'S comprising of more than 1500MB of data. Requirements from PGS_AA_dcw were to provide land/sea/ice flags for the world, so the relevant coverage from the data base was extracted; namely Political/Oceans. This coverage contains all the vector information pertaining to political boundaries and those which exist between certain cover types i.e., land/sea/ice. *(DCW states that the representation of international boundaries is not authoritative)*

The structure of the DCW database is represented in Figure D-1. The DCW database implements three types of VPF files: directories, tables and indices. The data base files are contained within a hierarchy of directories. Contained within these directories are the tables and indices that provide information. Each table within the database consists of two parts the header and the data records. By examining the header, it is possible to locate the information wanted.

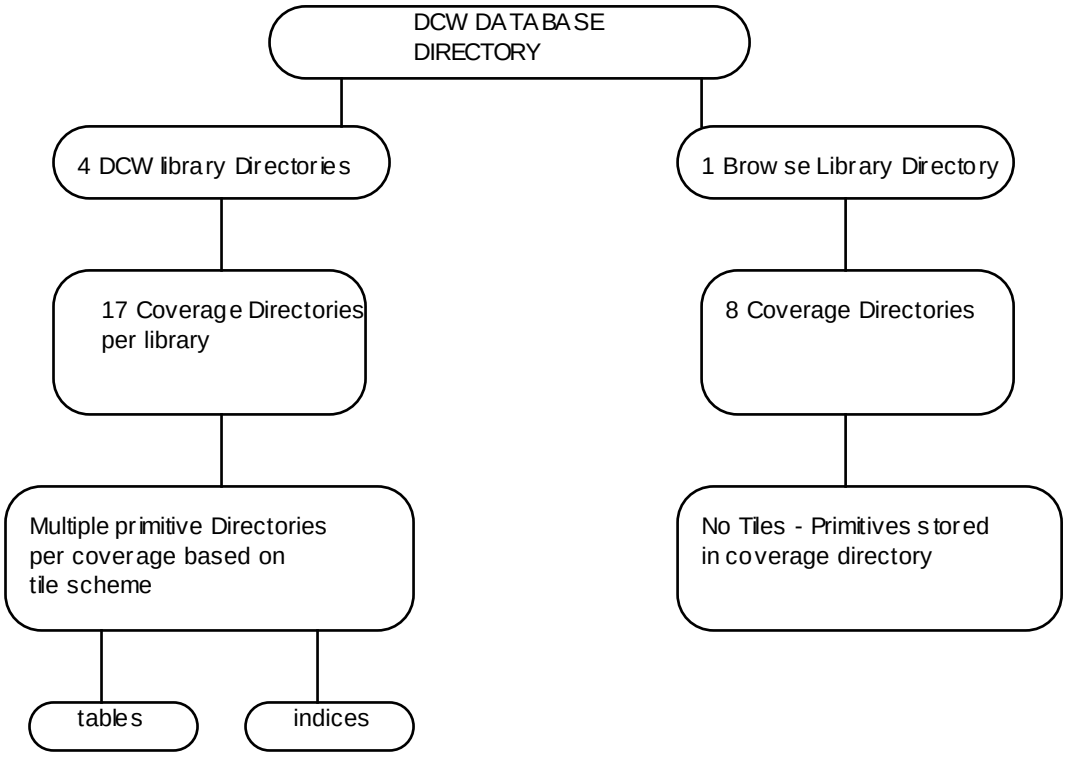


Figure D-1. DCW Database Directory

D.2.2 Outline Functionality

D.2.2.1 Outline

As the tool design is at present, the inputs needed to extract land/sea cover flags are as follows.

a. The parameter name - at present only PO (Political / Oceans)

b. The number of parameters - at present only 1

c. The longitude of the point(s) - in the form +/- 180.0000; e.g. 134.2234

d. The latitude of the point(s) - in the form +/- 90.0000 e.g. 87.8945

e. The number of points - 1 or more

f. A results array already specified by the user. (This will be filled up by the tool)

E.G.

PGS_AA_dcw ('po', 1, 34.222, 87.8923, 1, [100][10]);

The tool looks at each **long/lat** pair in turn, and searches the database. The first hurdle the tool encounters is the set up of the DCW database. The world has been divided into four areas:

- Europe and northern Asia
- South America, Africa and Antarctica
- North America
- Southern Asia and Australia

To find the relevant location; and extract the data base value; the tool works in the following way.

a. Locate within which continent the search point lies

b. Locate the table containing the search point.

> **NOTE**: There may be cases within the Database where the point lies on the junction of two edges; and because of machine accuracy and scale issues, the database will provide no return to the search. If this happens the search is performed again with the addition of a value that will not alter the search due to scalar issues, but will move the point away from the junction so a value can be extracted.

c. Open the relevant table.

d. Locate the search point within the table.

e. Extract the value pertaining to that search point.

f. Close the table.

g. Return the result of the search.

h. Perform another search using the next input coordinate pair.

D.2.3 Optimal Operation

Optimal operation for extraction of data from the data base is accomplished at present by running the tool as stated above. The tool can be run in two modes. The first is calling the tool with one point at a time, the second being calling the tool once with all the points needed as inputs. Of the two the latter is the fastest.

D.2.4 Upgrades

D.2.4.1 Access Speed

At present the tool goes through the above process for every location, provided by the user, as can be expected this will slow down the search process and tool performance. There is a mechanism by which the tool can be speeded up - which may be implemented at a later date, and involves using the file headers in a more constructive fashion. Within the header, there is information about the adjoining tiles. Since most users will be using this tool in a swath based format, the tool will become more time efficient by staying down at the table level, and utilizing code to extract adjoining tile identifiers - rather than performing the search criterion for every single search location.

D.2.4.2 Additional Coverages

The tool has been developed in such a way, that if requirements for other coverages i.e., vegetation, drainage, hypsography are needed - all that is needed is for the data to be supplied, and an additional small code change made to facilitate the new parameters. The results array will then be filled up with integer values representing the vegetation, drainage, etc., type to found at the location provided by the user.

D.3 PGS_AA_dem, PGS_AA_2DRead, PGS_AA_2Dgeo, PGS_AA_3DRead, PGS_AA_3Dgeo

D.3.1 Data Sets Accessed

D.3.1.1 Introduction

These tools are designed to give access to a wide range of data sets all having all of the following characteristics

- gridded (i.e. raster or cell structured), with parameter value or values associated with each cell constituting the substance of the data set.
- rectangular, having 2 or 3 dimensions
- formatted in simple binary or ASCII with (in C terms) char, float or double, short or long integers aligned to byte boundaries.
- the physical data set is sufficiently small to be loaded into machine memory.

The latter two of these points are involved with pre–processing and implementation issues respectively and are dealt with later (3.3.3.).

Several data sets have been delivered with the toolset. These data sets were considered useful for testing purposes and may also satisfy some science team requirements. They were obtained from NOAA's National Geophysical Data Center in Boulder, Colorado. They are described in outline below. Further details are found in the delivered format and support files (described below). Full details are found in the National Geophysical Data Center (NGDC) publications:

Global Ecosystems Database Version 1.0 (on CD–ROM) User's Guide EPA/600/R–92/194a

Global Ecosystems Database Version 1.0 (on CD–ROM) Documentation Manual (Disc–A) EPA/600/R–92/194b

Global View 4 CD–ROM set. United States Department of Commerce (USDC), National Oceanographic and Atmospheric Administration (NOAA), National Environmental Satellite Data and Information Service (NESDIS), National Geophysical Data Center (NGDC), Boulder Colorado.

Table D-1. Data Included in Toolkit 3/4/5 (1 of 2)

Data Set	Units	Cell size	File
Olson World Ecosystems v1.3a	30 cats	30 arc min	owe13a.img
Olson World Ecosystems v1.4d	74 cats	10 arc min	owe14d.img
Olson World Ecosystems v1.4dr	3 cats	10 arc min	owe14dr.img
Olson (Madagascar) Ecosystems v1.3a	29 cats	30 arc min	mowe13a.img
Federal Naval Operations Center (FNOC) modal elevation	Meters	10 arc min	fnocmod.imgs
FNOC maximum elevation	Meters	10 arc min	fnocmax.imgs
FNOC minimum elevation	Meters	10 arc min	fnocmin.imgs
FNOC modal elevation	Meters	10 arc min	fnocmod.img_dec
FNOC maximum elevation	Meters	10 arc min	fnocmax.img_dec
FNOC minimum elevation	Meters	10 arc min	fnocmin.img_dec
FNOC primary & 2ndary surface types	10 cats	10 arc min	fnocpt.img
FNOC ocean/land mask	2 cats	10 arc min	fnococm.img
FNOC number of ridges	Count	10 arc min	fnocrdg.img
FNOC direction of ridges	Degrees	10 arc min	fnocazm.img
FNOC water & urban cover	Percent	10 arc min	fnocwat.img
Zobler Soil types	108 cats	60 arc min	srzsoil.img
Zobler associated and included soil units	279 cats	60 arc min	srzsubs.img

Table D-1. Data Included in Toolkit 3/4/5 (2 of 2)

Data Set	Units	Cell size	File
Zobler associated and included soil units	279 cats	60 arc min	srzsubs.img_dec
Zobler near surface soil texture	10 cats	60 arc min	srztex.img
Zobler surface slope	10 cats	60 arc min	srzslop.img
Zobler soil phase	87 cats	60 arc mins	srzphas.img
Zobler special codes	12 cats	60 arc mins	srzcode.img
Zobler world areas	9 cats	60 arc mins	srzarea.img
Etop05 surface elevation	meters	5 arc mins	etop05.dat
Etop05 surface elevation	meters	5 arc mins	etop05.dat_dec
DMA conterminous USA	meters	30 arc secs	usatile.bin tiles (3)
Terrainbase global DEM (etop05 based)	meters	5 arc mins	tbase.bin
Terrainbase global DEM (etop05 based)	meters	5 arc mins	tbase .bin tiles (3)
Geoid	cm	15 arc mins	geoid.dat

Note 1. The _dec files are byte swapped to allow operation on DEC machines. PGS_AA_dem has a byte swapping utility built in which comes into operation on DEC machines.

Note 2. The table in section 3.2.2. Specifies the parameters names recognized by the tools.

Note 3. The tiled files are subdivided in order to reduce physical file size.

There is no loss of data. Access to the tiles will yield the same result as to the original whole data set.

These data sets are samples only. Other data sets may be delivered with later versions of the tool kit or the user may use his/her own data sets from other sources.

D.3.1.2 Support and Format Files

Support and format files are required for each data set. There is one format file per data set but there is not necessarily a one–to–one mapping between data sets and support files since the same support file can be used for similar data sets. The association between these files is specified operationally in the indexFile see section 3.2.2.

The support files for the delivered data sets have been created by the tool developers although in the longer term it is anticipated that users will create their own data sets and support data. These files are simple label = value ASCII files containing a set of values required by the tools.

Format files use the Freeform data description language to describe data file formats. A subset of possible descriptions is accepted by the tool. Full details of Freeform, including format specifications can be found in the Freeform Tutorial accessible on the ftp server ftp.ngdc.noaa.gov under /pub. Freeform is also a component of the software of the tools. An outline of Freeform data description applicable to the ancillary tools is found below.

D.3.1.2.1 Support File

The support file is constructed using a label = value format and read using the PGS_AA_PeV tool described elsewhere. It contains various values which define the format of the output buffer containing parameter values returned to the user. For 2 and 3 dimensional data sets, there are mandatory fields that must exist in the support file. These are described below with an explanation of how each is derived.

cacheFormat1	the data type of the **output** to be produced by the tool (short, long, double or float). On machines (e.g. sgi IRIX64, dec_alpha) 'long' datatype is eight bytes long. In such cases instead of using 'long', 'int64' must be used.
cacheFormat2	the number of decimal places in the **output** to be produced by the tool (applicable to double and float only)
cacheFormatBytes	the number of bytes represented by the data type of the **output**
parmMemoryCache	the size in bytes of the parameter requested once changed to the output type. The volume of the parameter from the whole data set.
dataType	the data type of the output to be produced by the tool (short, long, double or float).
autoOperation	a composite integer value made up of operations that must be applied to the data during access (see section 3.2.3)
fileMemoryCache	the size in bytes of the data set file in its **input** format (see format file below)
maxLat	maximum latitude of data set
minLat	minimum latitude of data set
maxLong	maximum longitude of data set
minLong	minimum longitude of data set
xCells	the number of data set cells in the X (fastest changing) dimension
yCells	the number of data set cells in the Y (slower changing) dimension
zCells	the number of data set cells in the Z (slowest changing) dimension (set 0 for 2d data sets)
funcIndex	index for the interpolation routine to be used. Currently only linear interpolation is supported for which the index is 0.
swapBytes	'yes' to indicate byte swapping is required on the result buffer else 'no'. Used only by the PGS_AA_dem tool on dec machines for cases where the data files have originated on foreign machines.
note 1.	parmMemoryCache and fileMemoryCache must be => the appropriate size in bytes

note 2. The dimensions (*Cells*) must be matched with the storage form of the data set in terms of dimension ordering.

For some data sets, additional support information may be required. The tools will currently deal with the National Meteorological Center (NMC) Rapid Update Cycle (RUC) model products that are in a polar stereographic projection. Thus the following must be present in the support file.

lowerLeftLat	of the grid origin
lowerLeftLong	of the grid origin (in E coordinates)
meshLength	length in meters of the cell
gridOrientation	in E coordinates

D.3.1.2.2 Freeform Data Description

Freeform is able to deal with a number of format types. The data sets delivered with the tool kit are all have relatively simple binary formats described in the '.bfm' files.

e.g. fnocMod 1 2 short 0

- the first item is the parameter name as requested by the user
- the second and third values are the start and stop byte positions of the parameter
- the data type. On machines (e.g. sgi IRIX64, dec_alpha) 'long' datatype is eight bytes long. In such cases instead of using 'long', 'int64' must be used.
- the number of values after the decimal point for float/doubles

These files describe the **input** format of the data set; i.e., the format of the data set; c.f. the **output** format described in the support file that is the format of the buffer delivered to the user through the tool.

The parameter is described once and Freeform assumes the same byte pattern throughout the data file, whatever its size. A data set file may contain multiple parameters with different data types. However, Freeform does not allow multiple parameters to be band interleaved; i.e., multiple parameters must have values individually interleaved, e.g. the format file:

fnocMod 1 2 short 0

another_parm 3 6 float 1

will allow Freeform to ingest a data set having binary data (when viewed)

34 45.3 33 46.1 45 712.3 .....etc.

The extension .bfm tells Freeform that the file is in binary format. Other extensions are contents are available in Freeform although the ancillary tools will not deal with them at this release.

D.3.2 Functionality and Operation

D.3.2.1 Outline Functionality

The tools are designed to be called by the user using a parameter name; a file i.d.; an operation; a version number; and either geographic coordinates or file structure coordinates.

The tool takes the **parameter** name and matches it to a list in the indexFile. If found, the file i.d.s of the support and format files are ingested from the indexFile. The format and support files are then interrogated by the tool for relevant information. The **file i.d.** and **version** number provide the full identification for the data set file containing the parameter and must be known by the user from the process control environment (see 3.2.4.).

The **operation** is an integer comprising the sum of operations required by the user to be applied to the data during extraction through the tool. Section 3.2.3 specifies the available operations.

The **geographic coordinates** (input to **PGS_AA_2Dgeo, PGS_AA_3Dgeo**) are simple latitude/ longitude as double values in the range +/- 180.000 (longitude) and +/-90.000 (latitude). The **file structure coordinates** (x,y and z) (input to **PGS_AA_2DRead, PGS_AA_3DRead**) are defined in respect to the ordering of the data in the data set file. The calling sequence expects the 'x' dimension to be the fastest changing dimension followed by 'y' and (for 3 D data sets) 'z'. This means that the user must understand the nature of the ordering of dimensions in the data set file and this should also be reflected in the support file.

> **Example:**
>
> The Olson World Ecosystem Data sets supplied with the tool are ordered with lines of latitude first (i.e., the cells in the binary file start at +90.00, -180.00 and proceed to +90.00, +180.00 before starting the next line of latitude). Thus the value of longitude of each cell changes fastest and so longitude is the x dimension and latitude the y dimension.

Both GEO tools assume that longitude is associated with the fastest changing (x) dimension and perform calculations on this basis. This means the support file xCells value represents the longitude range of the data set. If a data set is oriented with latitude changing fastest then xCells must be set to the number of cells in latitude, and the latitude and longitude input arguments to the calling sequence must be used reversed in meaning; i.e., input user latitude into the longitude argument etc.

PGS_AA_dem operates in a very similar way to PGS_AA_2Dgeo that it utilizes. The value added in the DEM tool is that it selects parameter values from the same logical data set where the data are physically separated into tiles. The DEM tool makes the selection on the basis of the maxLat, maxLong, minLat, and minLong attributes found in the Support files.

D.3.2.2 Parameters and the indexFile

The AA tools have been delivered with a sample set of data files. These files contain one parameter only per data file, although the tools will operate with files having multiple parameters (with a limit currently set to 4). The indexFile currently contains parameters found in the sample

data sets. **The parameter names in the indexFile are those which must be used in the calling sequences.** When the user wishes to add new data sets, the indexFile must be updated with suitable names for the parameter(s) contained in the data sets plus the i.d.s of the support and format files (i.d.s should cross reference with process control table).

The current indexFile appears as follows:

Table D-2. Current Index File

Parameter 21 (Number Of Records)	Support File I.D.	Format File I.D.
OlsonMadegascarEcosystems1.3a	10901	10920
OlsonWorldEcosystems1.3a	10902	10921
OlsonWorldEcosystems1.4d	10903	10922
OlsonWorldEcosystems1.4dr	10903	10923
etop05SeaLevelElevM	10904	10924
fnocAzm	10905	10925
fnocOcm	10905	10926
fnocPt	10905	10927
fnocRdg	10905	10928
fnocSt	10905	10929
fnocUrb	10905	10930
fnocWat	10905	10931
fnocMax	10906	10932
fnocMin	10906	10933
fnocMod	10906	10934
srzArea	10907	10935
srzCode	10907	10936
srzPhas	10907	10937
srzSlop	10907	10938
srzSoil	10907	10939
srzText	10907	10940
nmcRucSigPres	10909	10941
nmcRucSigPot	10909	10941
usadmaelevation	10740 - 10751	10700 - 10711 (2)
tbaseElevationWorld	10915	10942
tbaseElevation	10916 - 10919	10943- 10946(2)
geoid data	10948	10947

Note 1: The nmc file contains 2 parameters of many from a model run for a test period. They are included for test purposes only and are not generally applicable.

Note 2: These elevation parameters cover multiple physical files that are accessed automatically by the DEM tool.

D.3.2.3 Use of User Specified and Auto–Operations

To account for the variability of data sets, two types of 'operation' have been enabled within the tools; user and auto–operations. The **user operation**, the last argument in the calling sequence,

specifies which additional functions the user wishes to apply to the data. The currently available operations are:

Operation: PGS_AA_NEARESTCELL

Argument value: 1

Applicable to: PGS_AA_2Dgeo, PGS_AA_3Dgeo

Function:

The geographic coordinates are translated to a column and row coordinate pair. The translation provides a floating point number. Obviously the cell coordinate is an integer. This operation allows the user to specify the nearest cell by rounding the floating point numbers up (using the C 'ceil' function).

Operation: PGS_AA_OP_NINTCELL

Argument value: 2

Applicable to: PGS_AA_3Dgeo

Function:

This operation is specific to the polar stereographic auto–operation the output from which is unclear at the boundary. This user operation is used to round geocoordinate values in a very similar way to PGS_AA_NEARESTCELL but with allowance for uncertain boundary calculations.

Operation: PGS_AA_INTERP2BY2

Argument value: 4

Applicable to: PGS_AA_2Dgeo

Function:

This operation conducts interpolation on a 2x2 grid (i.e. nearest 4 points) and returns the interpolated value. The type of interpolation is controlled by funcIndex defined in the support file. Currently only bilinear interpolation is supported with funcIndex = 0. The interpolation routine was taken from Numerical Recipes in C by William H. Press et al., pages 90 and 106.

Operation: PGS_AA_INTERP3BY3

Argument value: 8

Applicable to: PGS_AA_2Dgeo

Function:

This operation conducts interpolation on a 3x3 grid (i.e. nearest 9 points) and returns the interpolated value. The type of interpolation is controlled by funcIndex defined in the support file. Currently only bilinear interpolation is supported with funcIndex = 0. The interpolation routine was taken from Numerical Recipes in C by William H. Press et al., pages 90 and 106.

Other more complex operations can be conceived although none have been implemented at this time.

Auto–operations are those functions that must be applied in order to extract the correct values. The auto–operation is specified in the support file and applied automatically on each run. The currently available auto–operations are:

Operation: PGS_AA_AOP_PLATTECARRE
Support file value: 1
Applicable to: PGS_AA_2Dgeo, PGS_AA_3Dgeo, PGS_AA_dem
Function:

This auto–operation calculates the column row cell coordinates from geographic coordinates assuming a Platte Carre projection

Operation: PGS_AA_AOP_POLARSTEREO
Support file value: 2
Applicable to: PGS_AA_3Dgeo
Function:

This auto–operation calculates the column row cell coordinates from geographic coordinates assuming an NMC RUC model polar stereographic projection.

Operation: PGS_AA_AOP_GREENWICHSTART
Support file value: 4
Applicable to: PGS_AA_2DRead, PGS_AA_3DRead, PGS_AA_2Dgeo, PGS_AA_3Dgeo, PGS_AA_dem
Function:

This auto–operation recalculates the geographic coordinates assuming a longitude 0 value at Greenwich.

Operation: PGS_AA_AOP_IDLSTART
Support file value: 8
Applicable to: PGS_AA_2DRead, PGS_AA_3DRead, PGS_AA_2Dgeo, PGS_AA_3Dgeo, PGS_AA_dem
Function:

This auto–operation recalculates the geographic coordinates assuming a longitude 0 value at the Interactive Data Language (IDL).

Auto–operations are generally applied before user operations.

Both types of operation are additive; e.g., an auto–operation of value 9 will results in the functions PGS_AA_AOP_IDLSTART and PGS_AA_AOP_PLATTECARRE being applied to input geo–coordinates in that order.

D.3.2.4 Operational Environment

The file set i.d. and version number must be provided by the user to the ancillary tool. For a static data set, only the i.d. is relevant, the version number should be set to 1. The i.d. is set up in the process control table during Algorithm Integration and Test (AI&T) of the algorithm and should be known to the user.

For dynamically changing data sets, a version number is required which specifies the exact data file out of a number staged for the processing run (e.g., for a set of times). These are obtained from the process control tools PGS_PC_GetNumberOfFiles and PGS_PC_GetAttributes (described elsewhere in this document). The sequence from calling these tools to obtain a version number is:

PGS_PC_GetNumberOfFiles gets number of versions for a particular i.d.

LOOP FOR number of version with same file i.d.

PGS_PC_GetAttributes of each file version

test of attributes using user criterion

ENDLOOP

PGS_AA_tool call using i.d. and selected version number

This series of calls is the basis of the **PGS_AA_dem** tool that selects the correct tiles using geographic coverage attributes. DEMs or other 2 dimensional data sets that are physically too large to be ingested into RAM in one go, can be 'tiled' into smaller coverages. These are then entered into the PCF having the same fileId but different version numbers. The PGS_AA_dem tool makes the selection and fills the results buffer for the user.

D.3.3 Optimal Operation

D.3.3.1 Buffering

The tools ingest the whole data file into a buffer and then extracts the parameter required into a further parameter buffer. The area requested is then extracted and returned in the output/results buffer. The parameter buffer is "free'd" before exiting the tool. This leaves the file buffer in memory. Subsequent calls requesting parameter values from the same file are serviced from this buffer while parameters from other files obviously cause the new file to be buffered. There is a user configurable number of file buffers which can be held by each tool. It should be set by the user according to the memory limitation of the host machine and the need for rapid access. Obviously, the greater the number of files held, the quicker different parameter calls will be

serviced, but at the expense of tying up memory. The #define is currently set to 4 in PGS_AA.h (FORTRAN version is PGS_AA.f):

```
#define PGSd_AA_MAXNOCACHES 4
```

D.3.3.2 Multiple calls

The GEO tools can be used with single coordinate pairs repeatedly; e.g., calling the tool in a loop with changing lat/longs. The tools can also accept arrays of coordinate pairs. Using the tools in this way will illicit a much faster response from the tool since the setup functions called during each tool call are used only once.

D.3.3.3 Pre-processing, formats and file sizes

The static data files delivered with release 1 are in the format provided by the vendor. This format is compatible with Freeform since data set and Freeform development were associated at NGDC. Most of the files are of relatively small size and can readily be loaded into memory. Etop05 is somewhat larger (18 Mbytes) and especially when used with the FORTRAN interface, may demand memory that is not available (or only with virtual swapping).

The FORTRAN problem arises from the fact that only integers of type PGSt_integer which is equivalent to an Integer*4 are permitted. Thus PGS_AA_2DRead is forced to allocate, e.g., 36 Mbytes memory to extract the elevation data during a tool call. This is the principal reason behind tiling larger data sets such as the DEMs.

The ability of Freeform to deal with a range of formats means that pre–processing of many data sets should be minimal. However, data sets that are have a complex internal structure may require more extensive pre–processing. In particular, NMC data sets are multi–dimensional. It is not yet clear whether further tools will need to be developed to deal with these.

D.3.4 Setting up new/user data sets

Users can and are expected to use their own data sets. Below is a check list of the actions that need to be taken when introducing new data sets.

- Check that the data file conforms to the constraints outlined in 3.1.1.
- Construct a Freeform format file and a support file (3.1.2). Check that suitable operations are available and set the auto–operation.
- Edit a suitable file i.d. into the process control table for the data set, the format file and the support file. The latter 2 files must be in the support file section while the data set file i.d. must be in the product input section.
- Edit the indexFile to include a suitable parameter name for parameters in the data set (3.2.2). Include the file i.d.s of the format and support files related to the data set file and as inserted into the process control table.
- Place the data set file in the product input directory and the format and support files in the ~/runtime directory (or equivalent)

D.3.5 Upgrades

D.3.5.1 Interaction with HDF files

Where ancillary inputs are other EOS products, then the format from which the requested data must be extracted may be HDF. Further ancillary tools using HDF libraries may be developed to deal with this scenario.

D.3.5.2 Other format types for user files

Data sets that cannot be dealt with by the current tools may be due to having non–raster (e.g., vector) formats which may necessitate new tools; although possibly continuing to use Freeform. HDF libraries and formats may also be a means of accessing these formats.

D.3.5.3 New Operations

New data sets provided by ECS or the user may require new operations (user and/or auto). Where these are clearly defined and common to several processing chains, then the current tools may be upgraded to include new operations.

D.4 PGS_AA_PeVA

D.4.1 Data Sets accessed

PGS_AA_PeVA is an ancillary tool to be used for performing a parameter equals value extraction. There are three types of extraction that the tool can perform: a string, integer and a real from a parameter input.

The tool will only do this extraction from an ASCII file which the user constructs. An example of a file, is as follows:

CACHEFORMAT1 = long

CACHEFORMAT2 = 0

CACHEFORMATBYTES = 4

PARMMEMORYCACHE = 1036800

DATATYPE = long

DATARATE = static

ANEXAMPLEARRAY = (9,5,3,7)

D.4.2 Outline Functionality

The tool is designed to be called by the user, using a logical input, a parameter input and returning a value. The logical is an integer whose value is supplied through the PC environment, which gives the i.d. of the file to be acted upon by the PGS_AA_PeVA tool. The parameter is a

data set dependent character string produced by the user, and the value returned by the tool is the result of the mapping from the character string to its value.

Example of calling sequence to extract a string called MY_STRING from the logical file 10992, and return the resulting string in MY_STRING_VALUE.

PGS_AA_PeVA_string (10992, "MY_STRING", MY_STRING_VALUE);

The PGS_AA_PeVA tool operates in exactly the same way but allows for arrays to be extracted (see Main User Guide section)

D.4.3 Optimal Operation

There are some restrictions on the format of the data file. All parameter names must be in upper cases. Arrays must be formatted as shown in the example.

The PeV tool is based on Freeform, while the PeVA tool is based on ODL and will therefore produce different types of error conditions.

D.4.4 Upgrades

None anticipated.

This page intentionally left blank.

Appendix E. Example of Level 0 Access Tool Usage

This Appendix gives an end–to–end example of how Level 0 access tools might be used in science software.

As an example, we use CERES processing. The source document for TRMM formats is "Interface Control Document between the Sensor Data Processing Facility (SDPF) and the Tropical Rainfall Measuring Mission (TRMM) Customers," NASA Mission Operations and Data Systems Directorate, Draft, Nov. 1994. We assume that the TRMM mission specific parameters given in section 10 of that document apply to CERES.

A single normal CERES production run consists of 24 hours of data. For Level 0 processing, there is a single main instrument–specific science dataset, namely science telemetry (Application ID 54). There is also a "housekeeping" file, consisting of various APIDs, which is common to all TRMM instruments. All science data for one 24 hour period is contained in a single file; all other data, including calibration, diagnostic and housekeeping data are contained in a second file. In addition each of these datasets has an associated Detached SFDU (Standard Formatted Data Unit) Header file, which consists of TRMM file metadata.

E.1 Preparing Simulated CERES L0 Files

At the SCF, you must first prepare the input Level 0 data files. You may decide to customize your files by using function PGS_IO_L0_File_Sim in a C or FORTRAN program that you code yourself; alternatively you may choose to use the supplied interactive executable driver *L0sim*. The latter method is shown here. The sample given is for creating a science APID file. The housekeeping file generation inputs are slightly different

In the example,

data that you type is given like this;

data generated by program **L0sim** is given like this,

comments and explanations are given like this.

The line

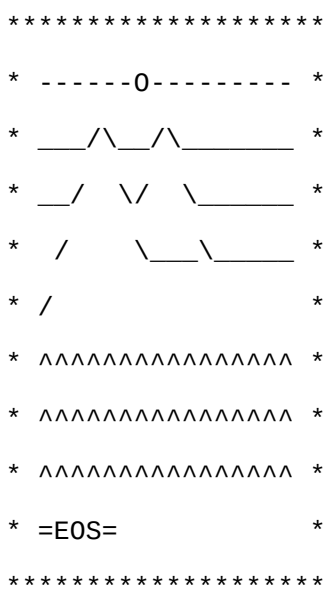
-->

means that you typed a carriage return, so using the default value.

unix% is the UNIX system prompt.

E.1.1 Sample Session

unix% *$PGSRUN/L0sim*



ECS L0 FILE SIMULATOR

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

enter spacecraft ID (TRMM, EOS_AM, EOS_AURA, EOS_PM_GIIS, EOS_PM_GIRD)
[TRMM]:
-->

enter start date in CCSDS ASCII (format A or B)
A) YYYY–MM–DDThh:mm:ss
B) YYYY–DDDThh:mm:ss

enter start date:
-->1997–12–01
enter stop date:
-->1997–12–02T00:00:00
You may leave out the entire time, minutes and seconds, or seconds if desired.

enter time interval in seconds [6.600000 sec]:
-->

enter the desired number of files [1]:
-->

TRMM always has only one file per APID (or housekeeping): EOS AM, PM and AURA may have more. Note that you must rerun program *L0sim* for each virtual data set you want, i.e., each "science" APID (or housekeeping); this prompt is asking how many files you want for a given virtual data set.

is this Housekeeping data (y/[n]):
-->

Housekeeping files are special in that they may have many APIDs. If you enter *y* here, you are prompted for the number of APIDs, then APID no. and data length for each APID. In this prototype, APIDs are written APID 1, APID 2, ..., APID n, APID 1, APID 2, ... until the stop time you requested is reached.

is this Quicklook data (y/[n]):
-->

For TRMM, the only effect of this input is to set a byte in the file header. For EOS AM, PM and AURA, there is no Quicklook data.

enter the Application ID [0]:
-->54

The APID is stamped on each packet. It is also written to the TRMM file header.

enter the Application Data Length [0]:
-->7118

This is the actual length of the packet application data in bytes. It does not include the packet header. All packets for a given APID have the same length.

read in Application Data from file [<none>]:
-->

If you type in the name of a file here, the simulator reads data from this file and writes it into the packet as application data. Here bytes 1–7118 of this file would be written to packet #1, bytes 7119–14238 to packet #2, etc.

specify processing options (y/[n]):

-->

This is for simulating some miscellaneous data in the TRMM file header. It is meant to indicate options applied during SDPF processing, before it gets to ECS.

start date: 1997–12–01T00:00:00
stop date: 1997–12–02T00:00:00
time interval: 6.6000 seconds

This will create approximately 94.65 MB of data.
accept ([y]/n)?
-->
Writing packets out to 1 file:

– start time of next file: 1997–12–01T00:00:00.000000Z
– number of packets in next file: 13091

– writing file: TRMM_G001_1997–12–01T00:00:00Z_V01.DATASET_01 ...
– writing files: TRMM_G001_1997–12–01T00:00:00Z_V01.DATASET_01 ...
TRMM_G001_1997–12–01T00:00:00Z_V01.SFDU_01

The SFDU file is only created for TRMM.

unix%

E.2 CERES Level 0 processing code using the SDP Toolkit

In this section is given an abbreviated example of what CERES L0 processing code might look like. It is assumed here that the datasets will be opened and processed one–at–a–time; this may not be the case in the actual CERES processing. No processing of packet, header or footer data returned is done in this example.

E.2.1 Notes

The examples show one way of retrieving simulated ephemeris and attitude data corresponding to packet times. For the science file (APID 54), the time of each packet is saved, then later used as input to the Toolkit ephemeris/attitude retrieval tool. To do this, a simulated ephemeris file must have been prepared beforehand. See the Toolkit Primer (Section 7) or Users Guide (Section 6.2.6) for details. (In the production system, this file is assumed to have been created in preprocessing from either Flight Dynamics Facility (FDF) files or from S/C ephemeris packets.)

In the interests of brevity, Detached SFDU Header file processing is completely omitted from the examples, as it is not clear what the information would be used for. Reading and accessing these files would involve use of the tools PGS_PC_GetFileAttr and PGS_PC_GetFileByAttr; see the Toolkit Primer (Section 4) for explanations of these.

Also, to keep things short, no error processing is shown.

The example code is given for illustrative purposes only, and is adapted from an unofficial unit test driver. The code given here has not actually been compiled and tested.

Because there is exactly one physical file per APID (or housekeeping) per day in TRMM L0 data, a virtual data set in Toolkit L0 functions corresponds to a single physical TRMM L0 file. For EOS AM, PM and AURA, there may be more than one physical file per given APID; in that case, this code would change, in that one must loop around the GetHeader and GetPacket calls until all physical files are read. There is an example of this in the tool descriptions for these two tools in section 6.2.1.1.

The examples assume the following exists in the PRODUCT INPUT FILES section of the Process Control File (PCF) at the SCF:

```
1|TRMM_G0001_1997-12-01T00:00:00Z_V01.dataset_01||||
                    TRMM_G0001_1997-12-01T00:00:00Z_V01.sfdu_01|1
```

```
      54|TRMM_G0088_1997-12-01T00:00:00Z_V01.dataset_01||||
                          TRMM_G0088_1997-12-01T00:00:00Z_V01.sfdu_01|1
```

(Note: each entry must appear on one line in the actual PCF, and not be broken into two lines as shown here.)

C code example

```
#include <PGS_IO.h>
#include <PGS_TD.h>

/* File logicals corresponding to PCF entries
    arbitrarily use APID as file logical, or 1 for housekeeping */
#define HOUSEKEEPING 1
#define SCIENCE 54

 /* PACKET_BUFFER_MAX is the maximum possible size of a telemetry packet,
    including packet header. Note that the input to L0sim corresponding to
    this is "Application Data Length"; however, the latter does *not* include
    packet header. Since the packet header is 14 bytes for TRMM, we used the
    value 7118 for the "Application Data Length" field in constructing the
    simulated files above. */
#define PACKET_BUFFER_MAX 7132

/* HEADER_BUFFER_MAX is the maximum possible size of the TRMM file header.
    This number is 26 for EOS AM, PM and AURA, since those file headers have
    no variable length part. */
#define HEADER_BUFFER_MAX 556

/* FOOTER_BUFFER_MAX is the maximum possible size of the TRMM file "footer,"
    which consists of Quality and Accounting Capsule (QAC) and optionally
    Missing Data Unit List (MDUL). This number is a wild guess. */
#define FOOTER_BUFFER_MAX 100000

/* NUM_DATASETS is the number of virtual datasets to process.
   This includes the housekeeping file and the science file. */
#define NUM_DATASETS 2

/* MAX_PKTS is the maximum number of packets.
   Used for saving packet times and for ephemeris and attitude retrieval */
#define MAX_PKTS 14000

main( )

{
PGSt_PC_Logical   file_logical[NUM_DATASETS];
                                    /* Logical file ID for PCF */

PGSt_SMF_status   returnStatus;     /* Toolkit function return value */
```

```
PGSt_integer      i;                /* Virtual data set loop index */

PGSt_IO_L0_VirtualDataSet
                  virtual_file;     /* Virtual file handle */
PGSt_double       start_time;       /* Virtual data set start time */
PGSt_double       stop_time;        /* Virtual data set stop time */

char              asciiUTC_A[28];   /* time in UTC CCSDS ASCII A format */

PGSt_IO_L0_Header header_buffer[HEADER_BUFFER_MAX];
                                    /* Buffer for receiving header data */
PGSt_IO_L0_Header footer_buffer[FOOTER_BUFFER_MAX];
                                    /* Buffer for receiving footer data */

PGSt_integer      j;                /* Index */
PGSt_integer      offset;           /* Offset byte of packet time */

PGSt_scTime       file_time[2][8];  /* File time in PB5 format */
PGSt_double       jdUTC[2];         /* Time in UTC -- Julian date format */
PGSt_boolean      onLeap;           /* Leap second flag */

PGSt_integer      packet_count;     /* No. packets in this file */

PGSt_integer      qac_size;         /* Size of QAC data in bytes */
PGSt_integer      mdul_size;        /* Size of MDUL data in bytes */

PGSt_integer      p;                /* Packets read counter */
PGSt_integer      packet_loop_flag;
                             /* Flag for controlling packet read loop */

PGSt_IO_L0_Packet packet_buf[PACKET_BUFFER_MAX];
                                    /* Buffer for receiving packet data */

PGSt_integer      appID;            /* Application ID of this packet */
PGSt_integer      pkt_seq_count;    /* Sequence number of this packet */
PGSt_integer      pkt_len;          /* Length in bytes of this packet */

PGSt_scTime       pkt_time[MAX_PKTS][8];
                                    /* Packet time stamps */
PGSt_double       UTC_offset[MAX_PKTS];
                                    /* packet UTC offset in seconds */

char              asciiUTC_A_eph_start[28];
            /* start time of ephemeris data in UTC CCSDS ASCII A format */
PGSt_double       positionECI[MAX_PKTS][3];
                                    /* ECI position vectors (m) */
PGSt_double       velocityECI[MAX_PKTS][3];
                                    /* ECI velocity vectors (m/s) */
PGSt_double       ypr[MAX_PKTS][3]; /* Euler angles (yaw/pitch/roll) (rad) */
PGSt_double       yprRate[MAX_PKTS][3];
```

```
                                    /* Euler angle rates (rad/sec) */
PGSt_double       attitQuat[MAX_PKTS][4];
                                    /* Attitude quaternions */

/*****************************************************************************/
/* For each data set (housekeeping or "science" APID)
/*****************************************************************************/

file_logical[0] = HOUSEKEEPING;
file_logical[1] = SCIENCE;

for( i=0; i<NUM_DATASETS; i++)
{

/*****************************************************************************/
/* Call PGS_IO_L0_Open to get a virtual file handle,
/*     start and stop times of the available data
/*****************************************************************************/

   returnStatus = PGS_IO_L0_Open( file_logical[i], TRMM,
      &virtual_file, &start_time, &stop_time);

/*****************************************************************************/
/* Translate times to ASCII in case you want to print them out or do
/*   something similar
/*****************************************************************************/

   returnStatus = PGS_TD_TAItoUTC(start_time,asciiUTC_A);
   returnStatus = PGS_TD_TAItoUTC(stop_time,asciiUTC_A);

/*****************************************************************************/
/* Call PGS_IO_L0_SetStart to position the file pointer at 20 minutes after
/*   data start
/*****************************************************************************/

   returnStatus = PGS_IO_L0_SetStart( virtual_file, start_time+1200. );

/*****************************************************************************/
/* Call PGS_IO_L0_GetHeader to retrieve header and footer
/*         information from the physical file
/*****************************************************************************/

   returnStatus = PGS_IO_L0_GetHeader( virtual_file,
      HEADER_BUFFER_MAX, header_buffer,
      FOOTER_BUFFER_MAX, footer_buffer );

/*****************************************************************************/
/*   Unpack and/or save or process header data here
/*****************************************************************************/
```

```
/*
   Header buffer contents:
   Bytes  1- 2 : 6 bits spare, 10 bits S/C ID
   Bytes  3-11 : S/C clock start time (PB5 format)
   Byte     12 : spare
   Bytes 13-21 : S/C clock stop time (PB5 format)
   Byte     22 : spare
   Bytes 23-26 : No. packets in file
*/

/*
   Convert S/C time to ASCII, in case you want to print it
*/

   for(j=0;j<8;j++)
   {
      file_time[0][j] = header_buffer[ 2+j];  /* start */
      file_time[1][j] = header_buffer[12+j];  /* stop */
    }
   for(j=0;j<2;j++)
  {
      returnStatus = PGS_TD_PB5toUTCjd( file_time[j], jdUTC );
      if( returnStatus == PGSTD_N_LEAP_SEC_IGNORED)
      {
         onLeap = PGS_TRUE;
      }
      else
      {
         onLeap = PGS_FALSE;
      }
      PGS_TD_UTCjdtoUTC( jdUTC, onLeap, asciiUTC_A);
   }

/* Special notes for EOS AM, PM and AURA:
    (1) 9th byte of file header time is not used in EOS AM or PM or AURA time
        conversions in this prototype
 /*
   Convert no. packets in file to integer
*/

packet_count =
      header_buffer[25] + 256 * (
      header_buffer[24] + 256 * (
      header_buffer[23] + 256 * (
      header_buffer[22] )));
```

```
/*************************************************************************/
/*  Convert footer sizes to integer: quality (QAC) and missing (MDUL) data
/*     (TRMM only)
/*************************************************************************/

   qac_size =
      footer_buffer[3] + 256 * (
      footer_buffer[2] + 256 * (
      footer_buffer[1] + 256 * (
      footer_buffer[0] )));
   mdul_size =
      footer_buffer[4+qac_size+3] + 256 * (
      footer_buffer[4+qac_size+2] + 256 * (
      footer_buffer[4+qac_size+1] + 256 * (
      footer_buffer[4+qac_size  ] )));

/*************************************************************************/
/* Note: the simulator does *not* simulate the internal structure of the QAC
/*   and MDUL data
/*************************************************************************/

/*************************************************************************/
/*   While still packets to process in this file
/*************************************************************************/

   p = 0;
   packet_loop_flag = 1;
   while( packet_loop_flag )
   {

/*************************************************************************/
/*      Call PGS_IO_L0_GetPacket to read a single L0 packet
/*      If reached end of file, set flag to exit loop
/*************************************************************************/

      returnStatus = PGS_IO_L0_GetPacket(
                  virtual_file, PACKET_BUFFER_MAX, packet_buf );
      if ( ( returnStatus == PGSIO_M_L0_HEADER_CHANGED )
          || ( returnStatus == PGSIO_W_L0_END_OF_VIRTUAL_DS ) )
      {
          packet_loop_flag = 0;
      }

/*************************************************************************/
/*   Unpack and/or save or process packet data
/*************************************************************************/
```

```
/*
   Packet buffer contents   -- "unused" means not written by simulator
   Bytes  1- 2 : packetID   bits 0-2:   Version Number        -- unused
                            bit 3:      Type                  -- unused
                            bit 4:      Secondary Header Flag -- unused
                            bits 5-15:  Application Process ID
   Bytes  3- 4 : pktSeqCntl bits 0-1:   Sequence Flags       -- unused
                            bits 2-15:  Packet Sequence Count
   Bytes  5 -6 : pktLength              Packet Length
   Bytes  7-14 : timeStamp              packet S/C time stamp
*/

      appID = packet_buf[1] + 256 * packet_buf[0];
      pkt_seq_count = packet_buf[3] + 256 * packet_buf[2];
      pkt_len = packet_buf[5] + 256 * packet_buf[4];

/* If currently processing the science file (APID 54),

Store time stamps for later retrieval of spacecraft ephemeris

       NOTE: Packet time format is spacecraft platform dependent */

       offset = 6; /* 6 for EOS_AM, 7 for EOS_PM */
      if( i == 1)
      {
         for(j=0;j<8;j++)
         {
            pkt_time[p][j] = packet_buf[offset+j];
         }
      }

      p++;
   }     /* End while (packet_Loop_flag) */

/*************************************************************************/
/* Call PGS_IO_L0_Close to close the virtual data set
/*************************************************************************/

   returnStatus = PGS_IO_L0_Close(virtual_file);
/*************************************************************************/
/* If currently processing the science file (APID 54),
/*    Retrieve simulated S/C ephemeris and attitude at packet times
/*    from previously prepared ephemeris file
/*************************************************************************/

   if( i == 1)
   {
      returnStatus = PGS_TD_SCtime_to_UTC( TRMM, pkt_time, p, asciiUTC_A,
UTC_offset );
```

```
      returnStatus = PGS_EPH_EphemAttit( TRMM, asciiUTC_A, UTC_offset,
                           PGS_TRUE, PGS_TRUE, asciiUTC_A_eph_start,
                           positionECI, velocityECI, ypr, yprRate, attitQuat );
   }

/*****************************************************************************/
/* End for (each data set)
/*****************************************************************************/

}

}
```

FORTRAN code 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     NUM_DATASETS
      parameter   (NUM_DATASETS=2)

      integer           pgs_mem_calloc
      integer           pgs_io_l0_open
      integer           pgs_td_taitoutc
      integer           pgs_io_l0_setstart
      integer           pgs_io_l0_getheader
      integer           pgs_td_pb5toutcjd
      integer           pgs_td_utcjdtoutc
      integer           pgs_io_l0_getpacket
      integer           pgs_io_l0_close
      integer           pgs_td_sctime_to_utc
      integer           pgs_eph_ephemattit

      integer           file_logical(2)
      integer           i

      integer           returnstatus
      integer           virtual_file
      double            precision start_time
      double            precision stop_time

      character*27      asciiutc_a
```

```
      character*556     header_buffer
      character*100000  footer_buffer

      integer           j
      character*8       file_time(2)
      double precision  jdutc(2)
      integer           onleap

      integer           packet_count

      integer           qac_size
      integer           mdul_size

      integer           packet_loop_flag

      character*7132    packet_buf

      integer           appid
      integer           pkt_seq_count
      integer           pkt_len
      integer           offset

      character*8       pkt_time(14000)
      double precision  utc_offset(14000)

      character*27      asciiutc_a_eph_start

      double precision  eciposition(3,14000)
      double precision  ecivelocity(3,14000)
      double precision  ypr(3,14000)
      double precision  yprrate(3,14000)
      double precision  attitquat(4,14000)

C ***************************************************************************/
C  For each data set (housekeeping or science APID)
C ***************************************************************************/

      file_logical(1) = 1
      file_logical(2) = 54

      do 10 i=1,NUM_DATASETS

C ***************************************************************************/
C  Call pgs_io_l0_open to get a virtual file handle,
C      start and stop times of the available data
C ***************************************************************************/

   returnstatus = pgs_io_l0_open( file_logical(i), TRMM, virtual_file,
      start_time,
      stop_time)
```

```
C ***************************************************************************/
C  Translate times to ASCII in case you want to print them out or do something
C   similar
C ***************************************************************************/

         returnstatus = pgs_td_taitoutc(start_time,asciiutc_a)
         returnstatus = pgs_td_taitoutc(stop_time,asciiutc_a)

C ***************************************************************************/
C  Call pgs_io_l0_setstart to position the file pointer at 20 minutes after
C  data start
C ***************************************************************************/

         returnstatus = pgs_io_l0_setstart( virtual_file, start_time+1200. )

C ***************************************************************************/
C  Call pgs_io_l0_getheader to retrieve header and footer
C          information from the physical file
C ***************************************************************************/

         returnstatus = pgs_io_l0_getheader( virtual_file, 556, header_buffer,
            100000, footer_buffer )

C ***************************************************************************/
C    Unpack and/or save or process header data here
C ***************************************************************************/

C
C   Header buffer contents:
C   Bytes  1- 2 : 6 bits spare, 10 bits S/C ID
C   Bytes  3-11 : S/C clock start time (PB5 format)
C   Byte     12 : spare
C   Bytes 13-21 : S/C clock stop time (PB5 format)
C   Byte     22 : spare
C   Bytes 23-26 : No. packets in file
C

C    Convert S/C start and stop time to ASCII, in case you want to print it

         do 20 j=1,8
            file_time[1] = header_buffer(3:11)
            file_time[2] = header_buffer(13:21)
20       continue

         do 30 j=1,2
            returnstatus = pgs_td_pb5toutcjd( file_time(j), jdutc )
            if( returnstatus .eq. PGSTD_N_LEAP_SEC_IGNORED) then
               onLeap = PGS_TRUE
            else
```

```
               onLeap = PGS_FALSE
            end if
            pgs_td_utcjdtoutc( jdutc, onleap, asciiutc_a)
30       continue

C  Special notes for EOS AM, PM and AURA:
C    (1) 9th byte of file header time is not used in EOS AM or PM or AURA time
C        conversions in this prototype
C
C Convert no. packets in file to integer
C

         packet_count =
     .            header_buffer(26) + 256 * (
     .            header_buffer(25) + 256 * (
     .            header_buffer(24) + 256 * (
     .            header_buffer(23) )))

C ***************************************************************************/
C    Unpack footer sizes: quality (QAC) and missing (MDUL) data (TRMM only)
C ***************************************************************************/

         qac_size =
     .            footer_buffer(4) + 256 * (
     .            footer_buffer(3) + 256 * (
     .            footer_buffer(2) + 256 * (
     .            footer_buffer(1) )))
         mdul_size =
      .           footer_buffer(4+qac_size+4) + 256 * (
     .            footer_buffer(4+qac_size+3) + 256 * (
     .            footer_buffer(4+qac_size+2) + 256 * (
     .            footer_buffer(4+qac_size+1) )))

C ***************************************************************************/
C    Note: the simulator does *not* simulate the internal structure of the QAC
C      and MDUL data
C ***************************************************************************/

C ***************************************************************************/
C    While still packets to process in this file
C ***************************************************************************/

         p = 1
         packet_loop_flag = 1

         do while( packet_loop_flag .eq. 1 )

C ***************************************************************************/
C       Call PGS_IO_L0_GetPacket to read a single L0 packet
```

```
C       If reached end of file, set flag to exit loop
C ***********************************************************************/

            returnStatus = pgs_io_l0_getpacket( virtual_file, 7132, packet_buf
)

            if ( ( returnStatus .eq. PGSIO_M_L0_HEADER_CHANGED )
     .             .or. ( returnStatus .eq. PGSIO_W_L0_END_OF_VIRTUAL_DS ) )
then
                packet_loop_flag = 0
            end if

C ***********************************************************************/
C    Unpack and/or save or process packet data
C ***********************************************************************/

C
C   Packet buffer contents   -- "unused" means not written by simulator
C   Bytes  1- 2 : packetID   bits 0-2:   Version Number        -- unused
C                            bit 3:      Type                  -- unused
C                            bit 4:      Secondary Header Flag -- unused
C                            bits 5-15:  Application Process ID
C   Bytes  3- 4 : pktSeqCntl bits 0-1:   Sequence Flags        -- unused
C                            bits 2-15:  Packet Sequence Count
C   Bytes  5 -6 : pktLength              Packet Length
C   Bytes  7-14 : timeStamp              packet S/C time stamp
C

            appID = packet_buf(2) + 256 * packet_buf(1)
            pkt_seq_count = packet_buf(4) + 256 * packet_buf(3)
            pkt_len = packet_buf(6) + 256 * packet_buf(5)

C  If currently processing the science file (APID 54),

C     Store time stamps for later retrieval of spacecraft ephemeris

C    NOTE: Packet time format is spacecraft platform dependent

            if( i .eq. 2 ) then
                offset = 7
                  pkt_time(p) = packet_buf(offset:14)
40              offset
            end if

            p = p + 1

C  End while (packet_loop_flag)

         end do
```

```
C ************************************************************************/
C  Call PGS_IO_L0_Close to close the virtual data set
C ************************************************************************/

         returnstatus = pgs_io_l0_close(virtual_file)

C ************************************************************************/
C  If currently processing the science file (APID 54),
C     Retrieve simulated S/C ephemeris and attitude at packet times
C     from previously prepared ephemeris file
C ************************************************************************/

         if( i .eq. 2) then
            returnstatus = pgs_td_sctime_to_utc( TRMM, pkt_time, p,
               asciiutc_a, utc_offset )

            returnstatus = pgs_eph_ephemattit( TRMM, asciiutc_a, utc_offset,
     .                     PGS_TRUE, PGS_TRUE, asciiutc_a_eph_start,
     .                     positioneci, velocityeci, ypr, yprrate, attitquat )
         end if

C ************************************************************************/
C  End for (each data set)
C ************************************************************************/

10    continue
```

Appendix F. Level 0 File Formats

This Appendix gives the definition of file formats assumed in construction of the Level 0 access tools, **PGS_IO_L0_***, and the file simulator **L0sim**. See section 6.2.1.1.

Notes on table entries:

-- "Y" in the SIM? Column means that this value is simulated by the L0sim software; no entry means that the value is either 0 or garbage in the simulated file.

-- No entry in the BIT column means bits 1_8.

F.1 Tropical Rainfall Measuring Mission (TRMM) File Formats

The source document for the TRMM file format is "Interface Control Document between the Sensor Data Processing Facility (SDPF) and the Tropical Rainfall Measuring Mission (TRMM) Customers," NASA Mission Operations and Data Systems Directorate, Draft, Nov. 1994. We assume that the TRMM mission specific parameters given in section 10 of that document apply to CERES and LIS.

TRMM has 2 files associated with each "science" APID or housekeeping file; a detached SFDU header file, an ASCII text file consisting of file metadata, and the main data file.

F.1.1 TRMM Files Schematic

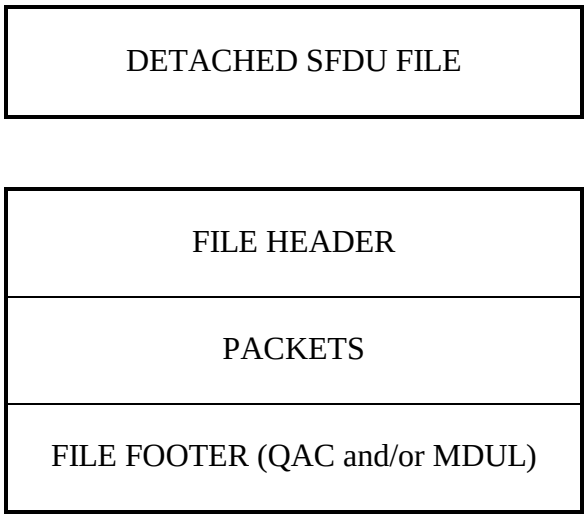


Figure F-1. TRMM Files Schematic

There is one pair of these files for each "science" APID, plus one pair for housekeeping. CERES has 3 "science" APIDs, thus will have 4 pairs of these files per day; LIS has one "science" APID, so will have 2 pairs per day.

F.1.2 Detached SFDU File

This is an ASCII text file containing file metadata. The format of this file is defined in the source document "Interface Control Document between the Sensor Data Processing Facility (SDPF) and the Tropical Rainfall Measuring Mission (TRMM) Customers," NASA Mission Operations and Data Systems Directorate, Draft, Nov. 1994, section 3.2.2.

Note: The Spacecraft Clock time format used in the file header is different from the format used for the packet Time Stamp.

F.1.3 TRMM File Header

Table F-1. TRMM File Header

Byte	Bit	Parameter	Sim?
1	1–6	(reserved)	
	7–8	Spacecraft ID	
2		Spacecraft ID	Y
3–11		Spacecraft Clock - first packet (PB5, microsec accuracy)	Y
12		(spare)	
13–21		Spacecraft Clock - last packet (PB5, microsec accuracy)	Y
22		(spare)	
23–26		Number of packets in file	Y
27		Processing Options	Y
28		Data Type Flag	Y
29–35		Time of Receipt at Originating Node (PB5, msec accuracy)	Y
36–38		(spare)	
39		Select Options	Y
40		Number of APIDs	Y
41–42		APID	Y
43		(spare)	
44		Number of QAC lists in File	Y
45–48		Offset to QAC list	Y

Byte numbers are shown for a "science" file.
Byte 2, Spacecraft ID, is always 6b (hex).
Byte 27, Processing Options:

bit 3 on, Redundant Data Deleted
bit 6 on, Data Merging
bit 7 in, RS Decoding

Byte 28, Data Type Flag:

=1, Routine Production Data
=2, Quicklook Data

Note: Routine production and quicklook files have the same format.

Bytes 29–35, Time of Receipt at Originating Node, is arbitrarily set to be equal to

Spacecraft Clock - last packet (without microseconds).

Byte 39, Select Options, is always 2, to indicate data organized by APID
Byte 40, Number of APIDs

=1, "Science" file
>1, Housekeeping file

Bytes 41–42 are repeated for each APID in a housekeeping file.
Byte 44, Number of QAC lists in File, is always 1.
Bytes 45–48, Offset to QAC list, is measured in bytes from the last byte of this

field to the QAC footer start. Equal to the total number of bytes in the packet data.

F.1.4 TRMM Packet Data

The source document for the TRMM packet data format is "Tropical Rainfall Measuring Mission (TRMM) Telemetry and Command Handbook,Ó TRMM_490_137, February 21, 1994.

Bytes 1–6 are known as the Primary Packet Header; bytes 7–14 are called the Secondary Packet Header.

Table F-2. TRMM Packet Data

Byte	Bit	Parameter	Sim?
1	1–3	Version Number	Y
	4	Type	Y
	5	Secondary Header Flag	Y
	6–8	Application Process ID (APID)	Y
2		Application Process ID (APID)	Y
3	1–2	Sequence Flags	
	3–8	Packet Sequence Count	Y
4		Packet Sequence Count	Y
5–6		Packet Length in bytes (=p)	Y
7–14		Time Stamp	Y
15-p+14		Application Data	Y

Byte 1, bits 1–3, Version Number, is always 000.
Byte 1, bit 4, Type, is always 0.
Byte 1, bit 5, Secondary Header Flag, is always 1.
Bytes 5–6, Packet Length, is defined as "the length of the entire packet, in bytes,

> less the length of the primary packet header [6 bytes],
> less one byte." This is equivalent to the length of the secondary packet header (8 for TRMM) + the length of the application data - 1,

F.1.5 TRMM File Footer

Table F-3. TRMM File Footer Table

Byte	Bit	Parameter	Sim?
1–4		QAC List Length in bytes (=q)	Y
5-q+4		QAC entries	
q+5-q+8		Missing Data Unit List Length in bytes (=m)	Y
q+9-q+m+8		Missing Data Unit (MDU) entries	

QAC and MDU entries are neither simulated nor read in this prototype.

There is no Missing Data Unit List (MDUL) in housekeeping files.

F.2 EOS AM File Formats

F.2.1 EOS AM File Schematic

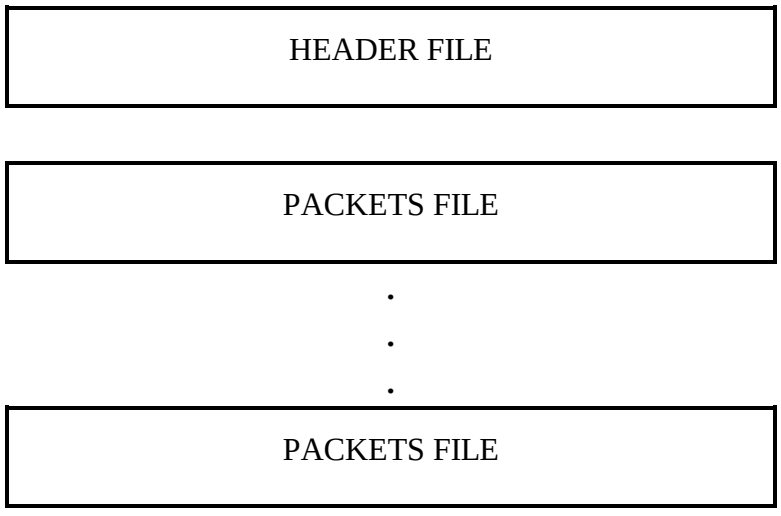


Figure F-2. EOS AM File Schematic

F.2.2 EOS AM File Header

EOS AM L0 data is contained in two or more files: a single header file (Construction Record) and one or more files containing packet data. The actual packet data files have no file header.

For a full description of the EOS AM file header see Interface Control Document Between The Earth Observing System (EOS) Data and Operations System (EDOS) ant the EOS Ground System (EGS) Elements (510-ICD-EDOS/EGS, CDRL B301), Mission Operations and Data Systems Directorate, Goddard Space Flight Center, November 5, 1999.

F.2.3 EOS AM Packet Data

Bytes 1–6 are known as the Primary Packet Header; bytes 7–15 are called the Secondary Packet Header.

Table F-4. EOS AM Packet Data

Byte	Bit	Parameter	Sim?
1	1–3	Version Number	Y
	4	Type	Y
	5	Secondary Header Flag	Y
	6–8	Application Process ID (APID)	Y
2		Application Process ID (APID)	Y
3	1–2	Sequence Flags	
	3–8	Packet Sequence Count	Y
4		Packet Sequence Count	Y
5–6		Packet Length in bytes (=p)	Y
7	1	Secondary Header ID Flag	Y
7	2–8	Time Stamp	Y
8–14		Time Stamp	Y
15	1	Quicklook Flag	
15	2–8	User Flags	
16-p+15		Application Data	Y

Byte 1, bits 1–3, Version Number, is always 000.
Byte 1, bit 4, Type, is always 0.
Byte 1, bit 5, Secondary Header Flag, is always 1.
Bytes 5–6, Packet Length, is defined as "the length of the entire packet, in bytes,

> less the length of the primary packet header [6 bytes],
> less one byte". This is equivalent to the length of the secondary packet
> header (9 for EOS AM) + the length of the application data - 1,

Byte 7, bit 1, Secondary header ID Flag, is always 0.
Byte 15, bit 1, Quicklook flag: EOS AM quicklook data has been eliminated by NASA.

There is no footer in EOS AM files.

F.3 EOS PM File Formats

F.3.1 EOS PM File Schematic

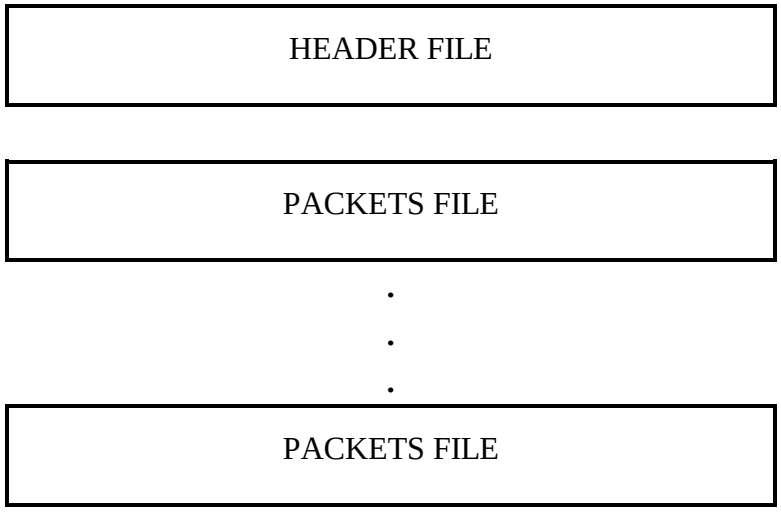


Figure F-3. EOS PM File Schematic

F.3.2 EOS PM File Header

EOS PM L0 data is contained in two or more files: a single header file (Construction Record) and one or more files containing packet data. The actual packet data files have no file header.

EOS PM file header for both GIRD and GIIS time formats is the same as the file header for EOS AM.

For a full description of the EOS PM file header see Interface Control Document Between The Earth Observing System (EOS) Data and Operations System (EDOS) ant the EOS Ground System (EGS) Elements (510-ICD-EDOS/EGS, CDRL B301), Mission Operations and Data Systems Directorate, Goddard Space Flight Center, November 5, 1999.

F.3.3 EOS PM Packet Data for GIIS Time Format

Bytes 1–6 are known as the Primary Packet Header; bytes 7–15 are called the Secondary Packet Header.

Table F-5. EOS PM GIIS Packet Data

Byte	Bit	Parameter	Sim?
1	1–3	Version Number	Y
	4	Type	Y
	5	Secondary Header Flag	Y
	6–8	Application Process ID (APID)	Y
2		Application Process ID (APID)	Y
3	1–2	Sequence Flags	
	3–8	Packet Sequence Count	Y
4		Packet Sequence Count	Y
5–6		Packet Length in bytes (=p)	Y
7	1	Secondary Header ID Flag	Y
7	2–8	Time Stamp	Y
8–14		Time Stamp	Y
15	1	Quicklook Flag	
15	2–8	User Flags	
16-p+15		Application Data	Y

Byte 1, bits 1–3, Version Number, is always 000.
Byte 1, bit 4, Type, is always 0.
Byte 1, bit 5, Secondary Header Flag, is always 1.
Bytes 5–6, Packet Length, is defined as "the length of the entire packet, in bytes,

> less the length of the primary packet header [6 bytes],
> less one byte". This is equivalent to the length of the secondary packet header (9 for EOS PM) + the length of the application data - 1,

Byte 7, bit 1, Secondary header ID Flag, is always 0.
Byte 15, bit 1, Quicklook flag: EOS PM quicklook data has been eliminated by NASA.

There is no footer in EOS PM files.

F.3.4 EOS PM Packet Data for GIRD Time Format

The source document for the EOS PM packe t data format is the Interface Control Document Between the Earth Observing System (EOS) Data and Operation System (EDOS) and the EOS Ground System (EGS) Elements (510-ICD-EDOS/EGS, CDPL B301), Mission Operations and Data System Directorate, Goddard Space Flight Center, November 5, 1999.

Bytes 1–6 are known as the Primary Packet Header; bytes 7–15 are called the Secondary Packet Header.

Table F-6. EOS PM GIRD Packet Data

Byte	Bit	Parameter	Sim?
1	1–3	Version Number	Y
	4	Type	Y
	5	Secondary Header Flag	Y
	6–8	Application Process ID (APID)	Y
2		Application Process ID (APID)	Y
3	1–2	Sequence Flags	
	3–8	Packet Sequence Count	Y
4		Packet Sequence Count	Y
5–6		Packet Length in bytes (=p)	Y
7	1	Secondary Header ID Flag	
7	2	Quicklook Flag	
7	3–8	User Flags	
8–15		Time Stamp	Y
16-p+15		Application Data	Y

Byte 1, bits 1–3, Version Number, is always 000.

Byte 1, bit 4, Type, is always 0.

Byte 1, bit 5, Secondary Header Flag, is always 1.

Byte 3, bits 1-2, Sequence Count = 11 (unsegmented).

Bytes 5-6 is Packet Data length = Number of Octets in the Data Zone minus 1.

Byte 7, bit 1, Secondary header ID Flag, is always 0.

Byte 7, bit 2, Quicklook flag, is always 0.

Byte 7, bits 3-8, User Flag is reserved for future instrument use.

Time Stamp = Expressed in CCSDS Unsegmented Time Code (CUC) where:

p-field = Bit 0 = second octet is present

Bits 1-3 = 010 = Epoch Time = Jan. 1, 1958

Bits 4-5 = 11 = 4 Octets Coarse Time Present

Bits 6-7 = 10 = 2 Octets Fine Time Present

p-field Extension = Bit 0 = 0 = No extension present

Bits 1-7 = Number of seconds to convert TAI to UTC.

T-field, Coarse = Bits 0-31 = Number of seconds since Jan. 1, 1958

T-field, Fine = Bits 0-15 = Sub-seconds time (LSB = 15.2 microseconds)

Bits 16-23 = fill bits (all zeros).

There is no footer in EOS PM files.

F.4 ADEOS-II File Formats

F.4.1 ADEOS-II File Schematic

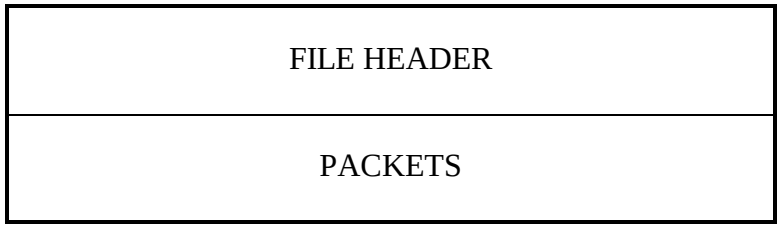


Figure F-4. ADEOS-II File Schematic

F.4.2 ADEOS-II File Header

Header format for ADEOS-II L0 files is unknown at this writing (Feb. 1995).

Arbitrarily we have taken the first 26 bytes of the TRMM file header as the EOS PM file header. Also, since the format of the Spacecraft Clock time in the file header is undefined, we arbitrarily take it as identical to the packet time stamp format.

Table F-7. ADEOS-II File Header

Byte	Bit	Parameter	Sim?
1	1–6	(reserved)	
	7–8	Spacecraft ID	
2		Spacecraft ID	
3–11		Spacecraft Clock - first packet	Y
12		(spare)	
13–21		Spacecraft Clock - last packet	Y
22		(spare)	
23–26		Number of packets in file	Y

F.4.3 ADEOS-II Packet Data

The ADEOS-II Packet Data format is preliminary and subject to change (as of 5/15/96).

The CHEM packet data format is the same as EOS PM GIRD Packet data format. Please refer to Section F.3. There is no ICD for this yet.

Bytes 1–6 are known as the Primary Packet Header; bytes 7–15 are called the Secondary Packet Header.

Table F-8. ADEOS-II Packet Data

Byte	Bit	Parameter	Sim?
1	1–3	Version Number	Y
	4	Type	Y
	5	Secondary Header Flag	Y
	6–8	Application Process ID (APID)	Y
2		Application Process ID (APID)	Y
3	1–2	Sequence Flags	
	3–8	Packet Sequence Count	Y
4		Packet Sequence Count	Y
5–6		Packet Length in bytes (=p)	Y
7-10		Instrument Time	Y
11		Pulse Time	Y
12-15		Orbit Time	Y
16-p+15		Application Data	Y

There is no footer in ADEOS-II files.

F.5 EOS AURA File Formats

The EOS AURA packet data format is the same as EOS PM GIRD packet data format. Please refer to Section F.3. There is no ICD at this time.

Appendix G. PGS_GCT Information Relating To Interface Specification

G.1 Projection Id's

PGSd_UTM (Universal Transverse Mercator)
PGSd_ALBERS (Albers Conical Equal Area)
PGSd_LAMCC (Lambert Conformal Conic)
PGSd_MERCAT (Mercator)
PGSd_PS (Polar Stereographic)
PGSd_POLYC (Polyconic)
PGSd_EQUIDC (Equidistant Conic)
PGSd_TM (Transverse Mercator)
PGSd_STEREO (Stereographic)
PGSd_LAMAZ (Lambert Azimuthal Equal Area)
PGSd_AZMEQD (Azimuthal Equidistant)
PGSd_GNOMON (Gnomonic)
PGSd_ORTHO (Orthographic)
PGSd_GVNSP (General Vertical Near–Side Perspective)
PGSd_SNSOID (Sinusoidal)
PGSd_EQRECT (Equirectangular)
PGSd_MILLER (Miller Cylindrical)
PGSd_VGRINT (Van der Grinten)
PGSd_HOM (Hotine Oblique Mercator--HOM)
PGSd_ROBIN (Robinson)
PGSd_SOM (Space Oblique Mercator--SOM)
PGSd_ALASKA (Modified Stereographic Conformal-- Alaska)
PGSd_GOOD (Interrupted Goode Homolosine)
PGSd_MOLL (Mollweide)
PGSd_IMOLL (Interrupted Mollweide)
PGSd_HAMMER (Hammer)
PGSd_WAGIV (Wagner IV)
PGSd_WAGVII (Wagner VII)
PGSd_OBLEQA (Oblated Equal Area)
PGSd_ISINUS and PGSd_ISINUS1 (Integerized Sinusoidal Grid)
PGSd_BCEA, PGSd_CEA (Cylindrical Equal_Area) (See Notes in section G.2.1)

G.1.1 NOTES

There have been some discrepancies in the output for SOM projection when used for satellites other than LANDSAT. Further investigations led us to the conclusion that the discrepancies were

due to a parameter called LANDSAT_RATIO used by the routines. It seemed that the gctpc routines were specifically designed to work for the Landsat satellites.

The documentation of GCTP software says that Landsat Ratio can be an input from the user through projection parameter. But, in fact in the GCTP source code this ratio has been hard coded for Landsat satellite which is 0.5201613.

This ratio causes the grid values to start near the North Pole instead of starting at equator at the ascending node. The explanation for this is as follows:

Landsat ratio 0.5201613 comes from the Landsat Scene calculations. It seems, in Landsat they divide each orbit into 248 Scenes. They want the starting point to be somewhere at the North Pole and they want it to start at Scene number 64.5 from the ascending node. This number when divided by the number of scenes for half of the globe which is 124 gives you 0.52016129. So by changing this ratio you are changing the start scene for the grid. Setting it to zero makes the grid values to start lets on the equator at the ascending node.

The LANDSAT_RATIO has been renames as satellite_ratio and the gctpc source code have been modified so that a user can now input the satellite ratio value through the projection parameters. For SOM option B, the satellite ratio is automatically set to 0.5201613.

G.2 GCTP Error Messages

If there is an error in the GCTP freeware library, the tools simply return PGSGCT_E_GCTP_ERROR. However, the actual errors are reported to the LogStatus file using the SMF interface. The list of possible GCTP errors are as follows:

Table G-1. GCTP Error Messages

Return	Description
PGSGCT_E_STD_PARALLEL	Equal latitudes for St. Parallels on opposite sides of equator
PGSGCT_E_ITER_EXCEEDED	Too many iterations in inverse
PGSGCT_E_POINT_PROJECT	Point projects into a circle of radius 2 * PI * radius_major
PGSGCT_E_INPUT_DATA_ERROR	Input data error
PGSGCT_E_STD_PARALLEL_OPP	Standard Parallels on opposite sides of equator
PGSGCT_E_INFINITE	Point projects into infinity
PGSGCT_E_ITER_FAILED	Iteration failed to converge
PGSGCT_E_PROJECT_FAILED	Point cannot be projected
PGSGCT_E_POINTS_ON_POLES	Transformation cannot be computed at the poles
PGSGCT_E_ITER_SOM	50 iterations without conv
PGSGCT_E_SPCS_ZONE	Illegal zone for the given spheroid
PGSGCT_E_SPCS_FILE	Error opening State Plane parameter file
PGSGCT_E_CONV_ERROR	Convergence Error
PGSGCT_E_LAT_15	Latitude failed to converge after 15 iterations
PGSGCT_E_LAT_CONVERGE	Latitude failed to converge

Table G-2. Projection Transformation Package Projection Parameters (1 of 2)

	Array Element							
Code & Projection Id	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1 PGSd_UTM	Smajor	Sminor						
2 PGSd_SPCS			Spheroid	Zone				
3 PGSd_ALBERS	Smajor	Sminor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
4 PGSd_LAMCC	Smajor	Sminor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
5 PGSd_MERCAT	Smajor	Sminor			CentMer	LtrueScale	FE	FN
6 PGSd_PS	Smajor	Sminor			LongPol	LtrueScale	FE	FN
7 PGSd_POLYC	Smajor	Sminor			CentMer	OriginLat	FE	FN
8 PGSd_EQUIDC (A)	Smajor	Sminor	STDPAR		CentMer	OriginLat	FE	FN
PGSd_EQUIDC (B)	Smajor	Sminor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
9 PGSd_TM	Smajor	Sminor	Factor		CentMer	OriginLat	FE	FN
10 PGSd_STEREO	Sphere				CentLon	CenterLat	FE	FN
11 PGSd_LAMAZ**	Smajor	Sminor			CentLon	CenterLat	FE	FN
PGSd_LAMAZ	Sphere				CentLon	CenterLat	FE	FN
12 PGSd_AZMEQD	Sphere				CentLon	CenterLat	FE	FN
13 PGSd_GNOMON	Sphere				entLon	CenterLat	FE	FN
14 PGSd_ORTHO	Sphere				CentLon	CenterLat	FE	FN
15 PGSd_GVNSP	Sphere		Height		CentLon	CenterLat	FE	FN
16 PGSd_SNSOID**	Smajor	Sminor			CentMer		FE	FN
PGSd_SNSOID	Sphere				CentMer		FE	FN
17 PGSd_EQRECT	Sphere				CentMer	LtrueScale	FE	FN
18 PGSd_MILLER	Sphere				CentMer		FE	FN
19 PGSd_VGRINT	Sphere				CentMer	OriginLat	FE	FN
20 PGSd_HOM (a)	Smajor	Sminor	Factor			OriginLat	FE	FN
PGSd_HOM (b)	Smajor	Sminor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
21 PGSd_ROBIN	Sphere				CentMer		FE	FN
22 PGSd_SOM (a)	Smajor	Sminor		IncAng	AscLong		FE	FN
PGSd_SOM (b)	Smajor	Sminor	Satnum	Path			FE	FN
23 PGSd_ALASKA	Smajor	Sminor					FE	FN
24 PGSd_GOOD	Sphere							
25 PGSd_MOLL	Sphere				CentMer		FE	FN
26 PGSd_IMOLL	Sphere							
27 PGSd_HAMMER	Sphere				CentMer		FE	FN
28 PGSd_WAGlV	Sphere				CentMer		FE	FN
29 PGSd_WAGVII	Sphere				CentMer		FE	FN
30 PGSd_OBEQA	Sphere		Shapem	Shapen	CentLon	CenterLat	FE	FN
31 PGSd_ISINUS1	Sphere				CentMer			
99 PGSd_ISINUS	Sphere				CentMer			
97 PGSd_CEA	Smajor	Sminor			CentMer	LtrueScale	FE	FN
98 PGSd_BCEA	Smajor	Sminor			CentMer	LtrueScale	FE	FN

***** With 5.2.19 release LAMAZ was generalized to support WGS84 ellipsoid in addition to the spherical Earth model. With 5.2.20 the same was applied to SINUSOIDAL projection.***

Table G-2. Projection Transformation Package Projection Parameters (2 of 2)

	Array Element				
Code & Projection Id	**9**	**10**	**11**	**12**	**13**
1 PGSd_UTM					
2 PGSd_SPCS					
3 PGSd_ALBERS					
4 PGSd_LAMCC					
5 PGSd_MERCAT					
6 PGSd_PS					
7 PGSd_POLYC					
8 PGSd_EQUIDC (A)	zero				
PGSd_EQUIDC (B)	one				
9 PGSd_TM					
10 PGSd_STEREO					
11 PGSd_LAMAZ					
PGSd_LAMAZ					
12 PGSd_AZMEQD					
13 PGSd_GNOMON					
14 PGSd_ORTHO					
15 PGSd_GVNSP					
16 PGSd_SNSOID					
17 PGSd_EQRECT					
18 PGSd_MILLER					
19 PGSd_VGRINT					
20 PGSd_HOM (a)	Long1	Lat1	Long2	Lat2	zero
PGSd_HOM (b)					one
21 PGSd_ROBIN					
22 PGSd_SOM (a)	PSRev	LRat	PFlag		zero
PGSd_SOM (b)					one
23 PGSd_ALASKA					
24 PGSd_GOOD					
25 PGSd_MOLL					
26 PGSd_IMOLL					
27 PGSd_HAMMER					
28 PGSd_WAGlV					
29 PGSd_WAGVII					
30 PGSd_OBEQA	Angle				
31 PGSd_ISINUS1	NZone		RFlag		
99 PGSd_ISINUS	NZone		RFlag		
97 PGSd_CEA					
98 PGSd_BCEA					

where

SMajor	Semi–major axis of ellipsoid
SMinor	Semi–minor axis of the ellipsoid
Spheroid	Used only for state plane projection. Use PGSd_CLARK66 (0) for 1927 datum or GRS80_WGS84(8) for 1983 datum
Sphere	Radius of reference sphere.
STDPAR	Latitude of the standard parallel
STDPR1	Latitude of the first standard parallel
STDPR2	Latitude of the second standard parallel
CentMer	Longitude of the central meridian
OriginLat	Latitude of the projection origin
FE	False easting in the same units as the semi–major axis
FN	False northing in the same units as the semi–major axis
LTrueScale	Latitude of true scale
LongPol	Longitude down below pole of map
Factor	Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator)
CentLon	Longitude of center of projection
CenterLat	Latitude of center of projection
Height	Height of perspective point
Long1	Longitude of first point on center line (Hotine Oblique Mercator, format A)
Long2	Longitude of second point on center line (Hotine Oblique Mercator, format A)
Lat1	Latitude of first point on center line (Hotine Oblique Mercator, format A)
Lat2	Latitude of second point on center line (Hotine Oblique Mercator, format A)
AziAng	Azimuth angle east of north of center line (Hotine Oblique Mercator, format B)
AzmthPt	Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B)
IncAng	Inclination of orbit at ascending node, counter–clockwise from equator (SOM, format A)
AscLong	Longitude of ascending orbit at equator (SOM, format A)
PSRev	Period of satellite revolution in minutes (SOM, format A)
LRat	Landsat ratio to compensate for confusion at northern end of orbit (SOM, format A -- For LANDSAT, use 0.5201613—See NOTES)
PFlag	End of path flag for Landsat: 0 = start of path, 1=end of path (SOM, format A)
Satnum	Landsat Satellite Number (1, 2, 3, 4 or 5, SOM format B)
Path	Landsat Path Number (Use WRS–1 (World Reference System) for Landsat 1, 2 and 3 and WRS–2 for Landsat4, 5 and 6.) (SOM, format B.) WRS–1 and WRS–2 can be found in Landsat User's Guide.
Nzone	Number of equally spaced latitudinal zones(rows); must be two or larger and even.
Rflag	Right justify columns flag is used to indicate what to do in zones with an odd of columns. If it has a value of 0 or 1, it indicates the extra column is on the right (zero) or left (one) of the projection Y-axis. If the flag is set to 2 (two),

the number of columns is calculated so there are always an even number of columns in each zone.

Shapem	Oblated Equal Area oval shape parameter m.
Shapen	Oblated Equal Area oval shape parameter n
angle	Oblated Equal Area oval rotation angle
zero	0
one	1

G.2.1 NOTES

Array elements 14 and 15 are set to zero
All array elements with blank fields are set to zero
All angles (latitudes, longitudes, azimuths, etc.) are in radians
Longitude is negative west of Greenwich
Latitude is negative south of equator

The following notes apply to the Space Oblique Mercator A projection.

A portion of Landsat rows 1 and 2 may also be seen as parts of rows 246 or 247. To place these locations at rows 246 or 247, set the end of path flag (parameter 11) to 1--end of path. This flag defaults to zero.

When Landsat - 1,2,3 orbits are being used, use the following values for specified parameters:

Parameter 4	99^{o} 5" 31.2' * PI/180 radians
Parameter 5	128.87 degrees - (360/251 * path number) * PI/180 radians
Parameter 9	103.2669323
Parameter 10	0.5201613

When Landsat–4,5 orbits are being used, use the following values for the specified parameters:

Parameter 4	99^{o} 12" 0' * PI/180 radians
Parameter 5	129.30 degrees - (360/233 * path number) * PI/180 radians
Parameter 9	98.884119
Parameter 10	0.5201613

*State plane projection is not included in this release. It will be included in the next release.

** The following notes apply for **BCEA and CEA projections,** and **EASE grid**:

HDFEOS 2.7 and 2.8, and SDPTK5.2.7 and 5.2.8:

Behrmann Cylindrical Equal-Area (BCEA) projection was used for 25 km global EASE grid. For this projection the Earth radius is set to 6371228.0m and latitude of true scale is 30 degrees. For 25 km global EASE grid the following apply:

```
Grid Dimensions:
      Width 1383
      Height 586
Map Origin:
      Column (r0) 691.0
      Row (S0) 292.5
      Latitude 0.0
      Longitude 0.0
Grid Extent:
      Minimum Latitude 86.72S
      Maximum Latitude 86.72N
      Minimum Longitude 180.00W
      Maximum Longitude 180.00E
      Actual grid cell size 25.067525km
```

Grid coordinates (r,s) start in the upper left corner at cell (0.0), with r increasing to the right and s increasing downward.

HDFEOS 2.9, SDPTK5.2.9 and later :

Although the projection code and name (tag) kept the same, BCEA projection was generalized to accept Latitude of True Scales other than 30 degrees, Central Meridian other than zero, and ellipsoid earth model besides the spherical one with user supplied radius. This generalization along with the removal of hard coded grid parameters will allow users not only subsetting, but also creating other grids besides the 25 km global EASE grid and having freedom to use different appropriate projection parameters. With the current version one can create the above mentioned 25 km global EASE grid of previous versions using:

```
Grid Dimensions:
      Width 1383
      Height 586
Grid Extent:
      UpLeft Latitude 86.72
      LowRight Latitude -86.72
      UpLeft Longitude -180.00
      LowRight Longitude 180.00
Projection Parameters:
      1) 6371.2280/25.067525 = 254.16263
      2) 6371.2280/25.067525 = 254.16263
      5) 0.0
      6) 30000000.0
      7) 691.0
      8) –292.5
```

Also one may create **12.5 km global EASE grid** using:

```
Grid Dimensions:
      Width 2766
      Height 1171
Grid Extent:
      UpLeft Latitude 85.95
      LowRight Latitude –85.95
      UpLeft Longitude –179.93
      LowRight Longitude 180.07
```

```
Projection Parameters:
      1) 6371.2280/(25.067525/2) = 508.325253
      2) 6371.2280/(25.067525/2) = 508.325253
      5) 0.0
      6) 30000000.0
      7) 1382.0
      8) –585.0
```

Any other grids (normalized pixel or not) with generalized BCEA projection can be created using appropriate grid corners, dimension sizes, and projection parameters. Please note that like other projections Semi-major and Semi-minor axes will default to Clarke 1866 values (in meters) if they are set to zero.

HDFEOS 2.10, SDPTK5.2.10 and later: A new projection CEA (97) was added to GCTP. This projection is the same as the generalized BCEA, except that the EASE grid produced will have its corners in meters rather than packed degrees, which is the case with EASE grid produced by BCEA.

HDFEOS 2.19, SDPTK5.2.19 and later: The Lambert Azimuthal Equal area projection was generalized to support WGS84 ellipsoidal Earth model in addition to the spherical model that was supported before. This generalization was needed to support EASE GRID 2.0 used for SMAP products.

HDFEOS 2.20, SDPTK5.2.20 and later: The Sinusoidal projection was generalized to support WGS84 ellipsoidal Earth model in addition to the spherical model that was supported before.

G.3 UTM Zone Codes

The Universal Transverse Mercator (UTM) Coordinate system uses zone codes instead of specific projection parameters. The table that follows lists UTM zone codes as used by GCTPc Projection Transformation Package. If southern zone is intended then use negative values.

Table G-3. Universal Transverse Mercator (UTM) Zone Codes

Zone	C.M.	Range	Zone	C.M.	Range
01	177W	180W–174W	31	003E	000E–006E
02	171W	174W–168W	32	009E	006E–012E
03	165W	168W–162W	33	015E	012E–018E
04	159W	162W–156W	34	021E	018E–024E
05	153W	156W–150W	35	027E	024E–030E
06	147W	150W–144W	36	033E	030E–036E
07	141W	144W–138W	37	039E	036E–042E
08	135W	138W–132W	38	045E	042E–048E
09	129W	132W–126W	39	051E	048E–054E
10	123W	126W–120W	40	057E	054E–060E
11	117W	120W–114W	41	063E	060E–066E

Table G-3. Universal Transverse Mercator (UTM) Zone Codes

Zone	C.M.	Range	Zone	C.M.	Range
12	111W	114W–108W	42	069E	066E–072E
13	105W	108W–102W	43	075E	072E–078E
14	099W	102W–096W	44	081E	078E–084E
15	093w	096W–090W	45	087E	084E–090E
16	087W	090W–084W	46	093E	090E–096E
17	081W	084W–078W	47	099E	096E–102E
18	075W	078W–072W	48	105E	102E–108E
19	069W	072W–066W	49	111E	108E–114E
20	063W	066W–060W	50	117E	114E–120E
21	057W	060W–054W	51	123E	120E–126E
22	051W	054W–048W	52	129E	126E–132E
23	045W	048W–042W	53	135E	132E–138E
24	039W	042W–036W	54	141E	138E–144E
25	033W	036W–030W	55	147E	144E–150E
26	027W	030W–024W	56	153E	150E–156E
27	021W	024W–018W	57	159E	156E–162E
28	015W	018W–012W	58	165E	162E–168E
29	009W	012W–006W	59	171E	168E–174E
30	003W	006W–000E	60	177E	174E–180W

Obtained from Software Documentation for GCTP general Cartographic Transformation Package: National Mapping Program Technical Instructions, U.S. Geological Survey, National Mapping Division, Oct. 1990,

Note: The following source contains UTM zones plotted on a world map:

Snyder, John P. *Map Projections--A Working Manual*; U.S. Geological Survey Professional Paper 1395

(Supersedes USGS Bulletin 1532), United States Government Printing Office, Washington D.C. 1987. p. 42.

State Plane Coordinate System uses zone codes instead of specific projection parameters. The table that follows lists State Plane Zone Codes as used by the GCTPc Projection Transformation Package.

Table G-4. State Plane Zone Codes (1 of 5)

Jurisdiction Zone name or number	NAD27 Zone Code	NAD83 Zone Code
Alabama		
East	0101	0101
West	0102	0102
Alaska		
01 through 10	5001	5001
thru	5010	5010
Arizona		
East	0201	0201
Central	0202	0202
West	0203	0203
Arkansas		
North	0301	0301
South	0302	0302
California		
01 through 07	0401	0401
thru	0407	0406
Colorado		
North	0501	0501
Central	0502	0502
South	0503	0503
Connecticut	0600	0600
Delaware	0700	0700
District of Columbia	1900	1900
Florida		
East	0901	0901
West	0902	0902
North	0903	0903

Table G-4. State Plane Zone Codes (2 of 5)

Jurisdiction Zone name or number	NAD27 Zone Code	NAD83 Zone Code
Georgia		
East	1001	1001
West	1002	1002
Hawaii		
01 through 05	5101	5101
thru	5105	5105
Idaho		
East	1101	1101
Central	1102	1102
West	1103	1103
Illinois		
East	1201	1201
West	1202	1202
Indiana		
East	1301	1301
West	1302	1302
Iowa		
North	1401	1401
South	1402	1402
Kansas		
North	1501	1501
South	1502	1502
Kentucky		
North	1601	1601
South	1602	1602
Louisiana		
North	1701	1701
South	1702	1702
Offshore	1703	1703
Maine		
East	1801	1801
West	1802	1802
Maryland	1900	1900
Massachusetts		
Mainland	2001	2001
Island	2002	2002
Michigan		
East (TM)	2101	----
Central (TM)	2102	----
West (TM)	2103	----
North (Lam)	2111	2111
Central (Lam)	2112	2112
South (Lam)	2113	2113

Table G-4. State Plane Zone Codes (3 of 5)

Jurisdiction Zone name or number	NAD27 Zone Code	NAD83 Zone Code
Minnesota		
North	2201	2201
Central	2202	2202
South	2203	2203
Mississippi		
East	2301	2301
West	2302	2302
Missouri		
East	2401	2401
Central	2402	2402
West	2403	2403
Montana	----	2500
North	2501	----
Central	2502	----
South	2503	----
Nebraska	----	2600
North	2601	----
South	2602	----
Nevada		
East	2701	2701
Central	2702	2702
West	2703	2703
New Hampshire	2800	2800
New Jersey	2900	2900
New Mexico		
East	3001	3001
Central	3002	3002
West	3003	3003
New York		
East	3101	3101
Central	3102	3102
West	3103	3103
Long Island	3104	3104
North Carolina	3200	3200
North Dakota		
North	3301	3301
South	3302	3302
Ohio		
North	3401	3401
South	3402	3402
Oklahoma		
North	3501	3501
South	3502	3502

Table G-4. State Plane Zone Codes (4 of 5)

Jurisdiction Zone name or number	NAD27 Zone Code	NAD83 Zone Code
Oregon		
North	3601	3601
South	3602	3602
Pennsylvania		
North	3701	3701
South	3702	3702
Rhode Island	3800	3800
South Carolina	----	3900
North	3901	----
South	3902	----
South Dakota		
North	4001	4001
South	4002	4002
Tennessee	4100	4100
Texas		
North	4201	4201
North Central	4202	4202
Central	4203	4203
South Central	4204	4204
South	4205	4205
Utah		
North	4301	4301
Central	4302	4302
South	4303	4303
Vermont	4400	4400
Virginia		
North	4501	4501
South	4502	4502
Washington		
North	4601	4601
South	4602	4602
West Virginia		
North	4701	4701
South	4702	4702
Wisconsin		
North	4801	4801
Central	4802	4802
South	4803	4803
Wyoming		
East	4901	4901
East Central	4902	4902
West Central	4903	4903
West	4904	4904

Table G-4. State Plane Zone Codes (5 of 5)

Jurisdiction Zone name or number	NAD27 Zone Code	NAD83 Zone Code
Puerto Rico	5201	5200
Virgin Islands	----	5200
St. John, St.	5201	----
Thomas	5202	----
St. Croix		
American Samoa	5300	----
Guam	5400	----

xxxfor converts input longitude and latitude to the corresponding x,y cartesian coordinates for the xxx projection. The following subroutines follow this general format:

utmfor (lon, lat, x, y) -- Universal Transverse Mercator (UTM)
stplnfor (lon, lat, x, y) -- State Plane
alberfor (lon, lat, x, y) -- Albers
lamccfor (lon, lat, x, y) -- Lambert Conformal Conic
merfor (lon, lat, x, y) -- Mercator
psfor (lon, lat, x, y) --Polar Stereographic
polyfor (lon, lat, x, y) --Polyconic
eqconfor (lon, lat, x, y) -- Equidistant Conic
tmfor (lon, lat, x, y) -- Transverse Mercator (TM)
sterfor (lon, lat, x, y) -- Stereographic
lamazfor (lon, lat, x, y) -- Lambert Azimuthal
azimfor (lon, lat, x, y) -- Azimuthal Equidistant
gnomfor (lon, lat, x, y) -- Gnomonic
orthfor (lon, lat, x, y) -- Orthographic
gvnspfor (lon, lat, x, y) -- General Near Side Perspective
sinfor (lon, lat, x, y) -- Sinusoidal
equifor (lon, lat, x, y) -- Equirectangular
millfor (lon, lat, x, y) -- Miller
vandgfor (lon, lat, x, y) -- Van Der Grinten
omerfor (lon, lat, x, y) -- Hotine Oblique Mercator (HOM)
robfor (lon, lat, x, y) -- Robinson
somfor (lon, lat, x, y) -- Space Oblique Mercator (SOM)
alconfor (lon, lat, x, y) -- Alaska Conformal
goodfor (lon, lat, x, y) -- Goode
molwfor (lon, lat, x, y) -- Mollweide
imolwfor (lon, lat, x, y) -- Interrupted Mollweide
hamfor (lon, lat, x, y) -- Hammer
wivfor (lon, lat, x, y) -- Wagner IV
wviifor (lon, lat, x, y) -- Wagner VII
obleqfor (lon, lat, x, y) -- Oblated Equal Area
isinusfor (lon, lat, x, y) – Integerized Sinusoidal Grid

bceafor (lon, lat, x, y) –Behrmann Cylindrical Equal-Area

xxxinv converts input x,y cartesian coordinates to the corresponding longitude and latitude for the xxx projection. The following subroutines follow this general format:

utminv(x, y, lon, lat) -- Universal Transverse Mercator (UTM)
stplninv(x, y, lon, lat) -- State Plane
alberinv(x, y, lon, lat) -- Albers
lamccinv(x, y, lon, lat) -- Lambert Conformal Conic
merinv(x, y, lon, lat) -- Mercator
psinv(x, y, lon, lat) -- Polar Stereographic
polyinv(x, y, lon, lat) -- Polyconic
eqconinv(x, y, lon, lat) -- Equidistant Conic
tminv(x, y, lon, lat) -- Transverse Mercator (TM)
sterinv(x, y, lon, lat) -- Stereographic
lamazinv(x, y, lon, lat) -- Lambert Azimuthal
aziminv(x, y, lon, lat) -- Azimuthal Equidistant
gnominv(x, y, lon, lat) -- Gnomonic
orthinv(x, y, lon, lat) -- Orthographic
gvnspinv(x, y, lon, lat) -- General Near Side Perspective
sininv(x, y, lon, lat) -- Sinusoidal
equiinv(x, y, lon, lat) -- Equirectangular
millinv(x, y, lon, lat) -- Miller
vandginv(x, y, lon, lat) -- Van Der Grinten
omerinv(x, y, lon, lat) -- Hotine Oblique Mercator (HOM)
robinv(x, y, lon, lat) -- Robinson
sominv(x, y, lon, lat) -- Space Oblique Mercator (SOM)
alconinv(x, y, lon, lat) -- Alaska Conformal
goodinv(x, y, lon, lat) -- Goode
molwinv(x, y, lon, lat) -- Mollweide
imolwinv(x, y, lon, lat) -- Interrupted Mollweide
haminv(x, y, lon, lat) -- Hammer
wivinv(x, y, lon, lat) -- Wagner IV
wviiinv(x, y, lon, lat) -- Wagner VII
obleqinv(x, y, lon, lat) -- Oblated Equal Area
isinusinv (x, y, lon, lat) – Integerized Sinusoidal Grid
bceainv (x,y,lon, lat) – Behrmann Cylindrical Equal-Area

This page intentionally left blank.

Appendix H. PGS_CUC_Cons - Example Standard Constants File

Current content of an Example standard constants file

Official file will be supplied by ESDIS Science Office

PI = 3.1415927

ATOMIC_SECOND = 9192631770

MOLECULAR_WEIGHT = 28.970

SOLAR_MOTION_VELOCITY = 19.7

PLANCKS_CONSTANT = 5.6697

This page intentionally left blank.

Appendix I. PGS_CUC_Conv—Input File Provided With the UdUnits Software

This tool uses the UdUnits package to provide unit conversions.

The following information taken from the input file provided with the UdUnits software describes the conversions currently available with the toolkit.

```
# $Id: udunits.dat,v 1.7 1994/02/03 17:20:02 steve Exp $
#
# The first column is the unit name. The second column indicates whether or
# not the unit name has a plural form (i.e., with an 's' appended).
# A 'P' indicates that the unit has a plural form, whereas, a 'S' indicates
# that the unit has a singular form only. The remainder of the line is the
# definition for the unit.
#
# '#' is the to–end–of–line comment–character.
#
# NB: When adding to this table, be *very* careful to distinguish between
# the letter 'O' and the numeral zero '0'. For example, the following two
# entries don't do what one might otherwise expect:
#
#       mercury_0C              mercury_32F
#       millimeter_Hg_0C        mm mercury_OC

#
# BASE UNITS. These must be first and are identified by a nil definition.
#
ampere                  P               # electric current
bit                     P               # unit of information
candela                 P               # luminous intensity
kelvin                  P               # thermodynamic temperature
kilogram                P               # mass
meter                   P               # length
mole                    P               # amount of substance
second                  P               # time
radian                  P               # plane angle

#
# CONSTANTS
#
percent         S 0.01
```

```
PI                  S 3.14159265358979323846
bakersdozen         S 13

%                   S percent
pi                  S PI

#
# NB: All subsequent definitions must be given in terms of
# earlier definitions. Forward referencing is not permitted.
#

#
# The following are non–base units of the fundamental quantities
#

#
# UNITS OF ELECTRIC CURRENT
#
A                   S ampere
amp                 P ampere
abampere            P 10 ampere             # exact
gilbert             P 7.957747e-1 ampere
statampere          P 3.335640e-10 ampere
biot                P 10 ampere

#
# UNITS OF LUMINOUS INTENSITY
#
cd                  S candela
candle              P candela

#
# UNITS OF THERMODYNAMIC TEMPERATURE
#
degree_Kelvin       P kelvin
degree_Celsius      S kelvin @ 273.15
degree_Rankine      P kelvin/1.8
degree_Fahrenheit   P degree_Rankine @ 459.67

#C                  S degree_Celsius        # `C' means `coulomb'
Celsius             S degree_Celsius
celsius             S degree_Celsius
centigrade          S degree_Celsius
degC                S degree_Celsius
degreeC             S degree_Celsius
degree_C            S degree_Celsius
degree_c            S degree_Celsius
```

```
deg_C                S degree_Celsius
deg_c                S degree_Celsius
degK                 S kelvin
degreeK              S kelvin
degree_K             S kelvin
degree_k             S kelvin
deg_K                S kelvin
deg_k                S kelvin
K                    S kelvin
Kelvin               P kelvin

degF                 S degree_Fahrenheit
degreeF              S degree_Fahrenheit
degree_F             S degree_Fahrenheit
degree_f             S degree_Fahrenheit
deg_F                S degree_Fahrenheit
deg_f                S degree_Fahrenheit
F                    S degree_Fahrenheit
Fahrenheit           P degree_Fahrenheit
fahrenheit           P degree_Fahrenheit
degR                 S degree_Rankine
degreeR              S degree_Rankine
degree_R             S degree_Rankine
degree_r             S degree_Rankine
deg_R                S degree_Rankine
deg_r                S degree_Rankine
#R                   S degree_Rankine     # `R' means `roentgen'
Rankine              P degree_Rankine
rankine              P degree_Rankine

#
# UNITS OF MASS
#
assay_ton            P 2.916667e2 kilogram
avoirdupois_ounce    P 2.834952e–2 kilogram
avoirdupois_pound    P 4.5359237e–1 kilogram  # exact
carat                P 2e–4 kilogram
grain                P 6.479891e–5 kilogram   # exact
gram                 P 1e–3 kilogram          # exact
kg                   S kilogram
long_hundredweight   P 5.080235e1 kilogram
metric_ton           P 1e3 kilogram           # exact
pennyweight          P 1.555174e–3 kilogram
short_hundredweight  P 4.535924e1 kilogram
slug                 P 14.59390 kilogram
```

```
troy_ounce          P 3.110348e–2 kilogram
troy_pound          P 3.732417e–1 kilogram
atomic_mass_unit    P 1.66044e–27 kilogram

tonne               P metric_ton
apothecary_ounce    P troy_ounce
apothecary_pound    P avoirdupois_pound
pound               P avoirdupois_pound
metricton           P metric_ton
gr                  S grain
scruple             P 20 grain
apdram              P 60 grain
apounce             P 480 grain
appound             P 5760 grain
atomicmassunit      P atomic_mass_unit
amu                 P atomic_mass_unit

t                   S tonne
lb                  P pound
bag                 P 94 pound
short_ton           P 2000 pound
long_ton            P 2240 pound

ton                 P short_ton
shortton            P short_ton
longton             P long_ton

#
# UNITS OF LENGTH
#
angstrom            P decinanometer
astronomical_unit   P 1.495979e11 meter
fathom              P 1.828804 meter
fermi               P 1e–15 meter           # exact
m                   S meter
metre               P meter
light_year          P 9.46055e15 meter
micron              P 1e–6 meter            # exact
mil                 P 2.54e–5 meter         # exact
nautical_mile       P 1.852000e3 meter      # exact
parsec              P 3.085678e16 meter
printers_pica       P 4.217518e–3 meter
printers_point      P 3.514598e–4 meter     # exact
US_statute_mile     P 1.609347e3 meter      # = intn'l mile + .000003 meter
US_survey_foot      P 3.048006e–1 meter
chain               P 2.011684e1 meter
```

```
inch                S 2.54 cm               # exact
astronomicalunit    P astronomical_unit
au                  S astronomical_unit
nmile               P nautical_mile
nmi                 S nautical_mile
inches              S inch
foot                S 12 inch               # exact
in                  S inch
barleycorn          P inch/3
ft                  S foot
feet                S foot
yard                P 3 foot
furlong             P 660 foot
international_mile  P 5280 foot             # exact
arpentlin           P 191.835 foot
yd                  S yard
rod                 P 5.5 yard
mile                P international_mile
arpentcan           P 27.52 mile

#
# UNITS OF AMOUNT OF SUBSTANCE
#
mol                 S mole

#
# UNITS OF TIME
#
day                 P 8.64e4 second         # exact
hour                P 3.6e3 second          # exact
minute              P 60 second             # exact
s                   S second
sec                 P second
shake               P 1e–8 second           # exact
sidereal_day        P 8.616409e4 second
sidereal_minute     P 5.983617e1 second
sidereal_second     P 0.9972696 second
sidereal_year       P 3.155815e7 second
tropical_year       P 3.155693e7 second
year                P 3.153600e7 second     # exact
eon                 P 1e9 year
d                   S day
min                 P minute
hr                  P hour
h                   S hour
```

```
fortnight       P 14 day
yr              P year
a               S year                  # "anno"

#
# UNITS OF PLANE ANGLE
#
#rad            P radian                # `rad' means `grey'
circle          P 2 pi radian
angular_degree  P (pi/180) radian
turn            P circle
degree          P angular_degree
degree_north    S angular_degree
degree_east     S angular_degree
degree_true     S angular_degree
arcdeg          P angular_degree
angular_minute  P angular_degree/60
angular_second  P angular_minute/60
grade           P 0.9 angular_degree    # exact
degrees_north   S degree_north
degreeN         S degree_north
degree_N        S degree_north
degreesN        S degree_north
degrees_N       S degree_north
degrees_east    S degree_east
degreeE         S degree_east
degree_E        S degree_east
degreesE        S degree_east
degrees_E       S degree_east
degree_west     S –1 degree_east
degrees_west    S degree_west
degreeW         S degree_west
degree_W        S degree_west
degreesW        S degree_west
degrees_W       S degree_west
degrees_true    S degree_true
degreeT         S degree_true
degree_T        S degree_true
degreesT        S degree_true
degrees_T       S degree_true
arcminute       P angular_minute
arcsecond       P angular_second
arcmin          P arcminute
arcsec          P arcsecond
```

```
#
# The following are derived units with special names. They are useful for
# defining other derived units.
#
steradian           P radian2
hertz               S 1/second
newton              P kilogram.meter/second2
coulomb             P ampere.second
lumen               P candela steradian
becquerel           P 1/second                  # SI unit of activity of a
#                                               # radionuclide
standard_free_fall  S 9.806650 meter/second2    # exact

pascal              P newton/meter2
joule               P newton.meter
hz                  S hertz
sr                  S steradian
force               S standard_free_fall
gravity             S standard_free_fall
free_fall           S standard_free_fall
lux                 S lumen/meter2
sphere              P 4 pi steradian
luxes               S lux
watt                P joule/second
gray                P joule/kilogram            # absorbed dose. derived unit
sievert             P joule/kilogram            # dose equivalent. derived unit
mercury_32F         S gravity 13595.065 kg/m3
mercury_60F         S gravity 13556.806 kg/m3
water_39F           S gravity 999.97226 kg/m3   # actually 39.2 F
water_60F           S gravity 999.00072 kg/m3
g                   S gravity
volt                P watt/ampere
mercury_0C          S mercury_32F
mercury             S mercury_32F
water               S water_39F
farad               P coulomb/volt
ohm                 P volt/ampere
siemens             S ampere/volt
weber               P volt.second
Hg                  S mercury
hg                  S mercury
H2O                 S water
h2o                 S water
tesla               P weber/meter2
henry               P weber/ampere
```

```
#
# The following are compound units: units whose definitions consist
# of two or more base units. They may now be defined in terms of the
# preceding units.
#

#
# ACCELERATION
#
gal                     P 1e–2 meter/second2 # exact

#
# Area
#
are                     P 1e2 m2                # exact
barn                    P 1e–28 m2              # exact
circular_mil            P 5.067075e–10 m2
darcy                   P 9.869233e–13 m2       # permeability of porous solids
hectare                 P 1e4 m2                # exact
acre                    P 4840 yard2

#
# ELECTRICITY AND MAGNETISM
#
abfarad                 P 1e9 farad             # exact
abhenry                 P 1e–9 henry            # exact
abmho                   P 1e9 siemens       # exact
abohm                   P 1e–9 ohm              # exact
abvolt                  P 1e–8 volt             # exact
C                       S coulomb
e                       S 1.6021917e–19 coulomb # charge of electron
chemical_faraday        P 9.64957e4 coulomb
physical_faraday        P 9.65219e4 coulomb
C12_faraday             P 9.64870e4 coulomb
gamma                   P 1e–9 tesla            # exact
gauss                   S 1e–4 tesla            # exact
H                       S henry
maxwell                 P 1e–8 weber            # exact
oersted                 P 7.957747e1 ampere/meter
S                       S siemens
statcoulomb             P 3.335640e–10 coulomb
statfarad               P 1.112650e–12 farad
stathenry               P 8.987554e11 henry
statmho                 P 1.112650e–12 siemens
statohm                 P 8.987554e11 ohm
statvolt                P 2.997925e2 volt
```

```
T                       S tesla
unit_pole               P 1.256637e–7 weber
V                       S volt
Wb                      S weber
mho                     P siemens
Oe                      S oersted
faraday                 P C12_faraday       # charge of 1 mole of
#                                                   # electrons

#
# ENERGY (INCLUDES WORK)
#
electronvolt            P 1.60219e–19 joule
erg                     P 1e–7 joule                # exact
IT_Btu                  P 1.055056 joule            # exact
EC_therm                P 1.05506e8 joule
thermochemical_calorie P 4.184000 joule             # exact
IT_calorie              P 4.1868 joule      # exact
J                       S joule
ton_TNT                 S 4.184e9 joule
US_therm                P 1.054804e8 joule          # exact
watthour                P watt hour
therm                   P US_therm
Wh                      S watthour
Btu                     P IT_Btu
calorie                 P IT_calorie
electron_volt           P electronvolt
thm                     S therm
cal                     S calorie
eV                      S electronvolt
bev                     S gigaelectron_volt

#
# FORCE
#
dyne                    P 1e–5 newton       # exact
pond                    P 1.806650e–3 newton        # exact
force_kilogram          S 9.806650 newton           # exact
force_ounce             S 2.780139e–1 newton
force_pound             S 4.4482216152605 newton    # exact
poundal                 P 1.382550e–1 newton
N                       S newton
gf                      S gram force
force_gram              P 1e–3 force_kilogram
force_ton               P 2000 force_pound          # exact
```

```
lbf                S force_pound
ounce_force        S force_ounce
kilogram_force     S force_kilogram
pound_force        S force_pound
ozf                S force_ounce
kgf                S force_kilogram
kip                P 1000 lbf
ton_force          S force_ton
gram_force         S force_gram

#
# HEAT
#
clo                P 1.55e–1 kelvin.meter2/watt

#
# LIGHT
#
lm                 S lumen
lx                 S lux
footcandle         P 1.076391e–1 lux
footlambert        P 3.426259 candela/meter2
lambert            P (1e4/PI) candela/meter2    # exact
stilb              P 1e4 candela/meter2         # exact
phot               P 1e4 lumen/meter2           # exact
nit                P 1 candela/meter2           # exact
langley            P 4.184000e4 joule/meter2    # exact
blondel            P candela/(pi meter2)

apostilb           P blondel
nt                 S nit
ph                 S phot
sb                 S stilb

#
# MASS PER UNIT LENGTH
#
denier             P 1.111111e–7 kilogram/meter
tex                P 1e–6 kilogram/meter        # exact

#
# MASS PER UNIT TIME (INCLUDES FLOW)
#
perm_0C            S 5.72135e–11 kg/(Pa.s.m2)
perm_23C           S 5.74525e–11 kg/(Pa.s.m2)
```

```
#
# POWER
#
voltampere              P volt ampere
VA                      S volt ampere
boiler_horsepower       P 9.80950e3 watt
shaft_horsepower        P 7.456999e2 watt
metric_horsepower       P 7.35499 watt
electric_horsepower     P 7.460000e2 watt       # exact
W                       S watt
water_horsepower        P 7.46043e2 watt
UK_horsepower           P 7.4570e2 watt
refrigeration_ton       P 12000 Btu/hour

horsepower              P shaft_horsepower
ton_of_refrigeration    P refrigeration_ton

hp                      S horsepower

#
# PRESSURE OR STRESS
#
bar                     P 1e5 pascal            # exact
standard_atmosphere     P 1.01325e5 pascal      # exact
technical_atmosphere    P 1 kg gravity/cm2      # exact
inch_H2O_39F            S inch water_39F
inch_H2O_60F            S inch water_60F
inch_Hg_32F             S inch mercury_32F
inch_Hg_60F             S inch mercury_60F
millimeter_Hg_0C        S mm mercury_0C
footH2O                 S foot water
cmHg                    S cm Hg
cmH2O                   S cm water
Pa                      S pascal
inch_Hg                 S inch Hg
inch_hg                 S inch Hg
inHg                    S inch Hg
in_Hg                   S inch Hg
in_hg                   S inch Hg
millimeter_Hg           S mm Hg
mmHg                    S mm Hg
mm_Hg                   S mm Hg
mm_hg                   S mm Hg
torr                    P mm Hg
foot_H2O                S foot water
ftH2O                   S foot water
```

```
psi                 S 1 pound gravity/in2
ksi                 S kip/in2
barie               P 0.1 newton/meter2
at                  S technical_atmosphere
atmosphere          P standard_atmosphere
atm                 P standard_atmosphere
barye               P barie

#
# RADIATION UNITS
#
Bq                  S becquerel
curie               P 3.7e10 becquerel          # exact
rem                 P 1e–2 sievert              # dose equivalent. exact
rad                 P 1e–2 gray                 # absorbed dose. exact
roentgen            P 2.58e–4 coulomb/kg        # exact
Sv                  S sievert
Gy                  S gray
Ci                  S curie
R                   S roentgen
rd                  S rad

#
# VELOCITY (INCLUDES SPEED)
#
c                   S 2.997925e+8 meter/sec
knot                P nautical_mile/hour

knot_international  S knot
international_knot  S knot
kt                  P knot

#
# VISCOSITY
#
poise               S 1e–1 pascal second        # absolute viscosity.
#                                               # exact
stokes              S 1e–4 meter2/second # exact
rhe                 S 10/(pascal second)        # exact
St                  S stokes

#
# VOLUME (INCLUDES CAPACITY)
#
acre_foot           S 1.233489e3 m3
board_foot          S 2.359737e–3 m3
bushel              P 3.523907e–2 m3
```

```
UK_liquid_gallon      P 4.546092e–3 m3
Canadian_liquid_gallon P 4.546090e–3 m3
US_dry_gallon         P 4.404884e–3 m3
US_liquid_gallon      P 3.785412e–3 m3
cc                    S cm3
liter                 P 1e–3 m3                  # exact. However, from 1901 to
#                                                # 1964, 1 liter = 1.000028 dm3
stere                 P 1 m3              # exact
register_ton          P 3.831685 m3
US_dry_quart          P US_dry_gallon/4
US_dry_pint           P US_dry_gallon/8
US_liquid_quart       P US_liquid_gallon/4
US_liquid_pint        P US_liquid_gallon/8
US_liquid_cup         P US_liquid_gallon/16
US_liquid_gill        P US_liquid_gallon/32
US_fluid_ounce        P US_liquid_gallon/128
US_liquid_ounce       P US_fluid_ounce
UK_liquid_quart       P UK_liquid_gallon/4
UK_liquid_pint        P UK_liquid_gallon/8
UK_liquid_cup         P UK_liquid_gallon/16
UK_liquid_gill        P UK_liquid_gallon/32
UK_fluid_ounce        P UK_liquid_gallon/160
UK_liquid_ounce       P UK_fluid_ounce
liquid_gallon         P US_liquid_gallon
fluid_ounce           P US_fluid_ounce
#liquid_gallon        P UK_liquid_gallon
#fluid_ounce          P UK_fluid_ounce
dry_quart             P US_dry_quart
dry_pint              P US_dry_pint
liquid_quart          P liquid_gallon/4
liquid_pint           P liquid_gallon/8
gallon                P liquid_gallon
barrel                P 42 US_liquid_gallon      # petroleum industry definition
quart                 P liquid_quart
pint                  P liquid_pint
cup                   P liquid_gallon/16
gill                  P liquid_gallon/32
tablespoon            P US_fluid_ounce/2
teaspoon              P tablespoon/3
peck                  P bushel/4
oz                    P fluid_ounce
floz                  S fluid_ounce
acre_feet             S acre_foot
board_feet            S board_foot
```

```
Tbl             P tablespoon
Tbsp            S tablespoon
tbsp            S tablespoon
Tblsp           S tablespoon
tblsp           S tablespoon
litre           P liter
l               S liter
tsp             S teaspoon
pk              S peck
bu              S bushel
fldr            S floz/8
dram            P floz/16
bbl             S barrel
pt              S pint
dr              S dram

#
# COMPUTERS AND COMMUNICATION
#
baud            S 1/second              # exact
b               S bit
bps             S bit/second
cps             S hertz
Bd              S baud

#
# MISC
#
kayser          P 1e2/meter             # exact
rps             S hertz
rpm             S hertz/60
geopotential    S gravity
work_year       P 2056 hours
work_month      P work_year/12

gp              S geopotential
dynamic         S geopotential
```

Appendix J. Population of Granule Level Metadata Using the SDP metadata tools

J.1 Introduction

The purpose of this appendix is to provide detailed guidance on the use of the SDP Toolkit for writing and reading granule-level metadata, i.e. the metadata that is associated with each instance of an input or output product. Section J.2 provides an overview of metadata in ECS and places the granule-level metadata handled by the toolkit in context with the larger metadata picture. Section J.3 outlines the procedures that are to be followed in interacting with ECS in the process of defining product metadata and provides a list of tools and references that will be useful in developing metadata. Section J.4 describes how metadata is generated and written to output files using the toolkit. Section J.4 also includes a discussion of how HDF and non-HDF product files are treated differently. Section J.5 discusses metadata toolkit usage. Section J.6 describes in detail the structure and syntax of the MCF. Section J.7 discusses metadata in HDF vs. non-HDF input and Output Files.

J.2 Overview of Metadata

Within ECS, the term "metadata" relates to all information of a descriptive nature that is associated with a product or dataset. This includes information that identifies a dataset, giving characteristics such as its origin, content, quality, and condition. Metadata can also provide information needed to decode, process and interpret the data, and can include items such as the software that was used to create the data.

These various types of information have been analyzed and developed into the ECS Earth Science Data Model, reference Document: 311-CD-008-001 (“Release B SDPS Database Design and Database Scheme Specifications”); and updates for the ECS Release 6A: 420-TP-022-001 (“Release 6A Implementation Earth Science Data Model”). Also see 420-TP-016-003 (“*Backus-Naur Format (BNF) Representation of the B.0 Earth Science Data Model for the ECS Project*”.)

J.2.1 The B.0 Earth Science Data Model

The ECS Data Model consists of a bounded set of attributes intended to cover the essential characteristics of all earth science data sets. This is sometimes referred to as “core” metadata. Not all core attributes are applicable to all data sets, but the core includes those attributes which most users employ to formulate searches and which most users would want to know about a data set when it was delivered.

All data or products in ECS belong to at least one collection. A collection is an aggregation of related elements called granules. A granule is the smallest piece of data that is independently managed by the system, i.e. represented by a record in the inventory. A granule may belong to more than one collection..

An ECS core metadata attribute can be collection-level, granule-level or both. Collection-level attributes describe a collection as a whole. These attributes include the collection name, the data center where the collection resides, the technical contact for the collection, etc. Granule-level attributes describe characteristics whose values vary granule to granule, such as the measurement time and location. If granule-level attributes are also present at the collection level, the collection-level attribute reflects the union of the values assigned to each granule. For example, a granule may have a start and stop time assigned to it. The collection-level start and stop times would be the earliest and latest times, respectively, of the member granules.

Individual collections may have important metadata associated with them which is not represented in the core set of metadata attributes. These are called *product-specific metadata,* and several options are available for handling them in ECS. Some product-specific metadata will reside in ECS database tables and will therefore be searchable by users, while other metadata will not. Whether product-specific metadata is searchable or not, and where and how it is supplied to the system is discussed in Section J.6.4.

J.2.2 Earth Science Data Types

Before a new collection can be added to ECS, an Earth Science Data Type (ESDT) descriptor file must be composed and submitted to Science Data Server, a component of the Data Server Subsystem. The ESDT descriptor file is parsed into components and used in various ECS subsystems as shown in Figure J-1. The ESDT descriptor file specifies the set of metadata attributes chosen to describe the collection. Most collection-level attributes are known beforehand so their values are specified in the descriptor file. Collection-level metadata attributes are delivered to the Interoperability Subsystem, which uses them to generate advertisements and entries for the GCMD, as well as the Data Management Subsystem, to support distributed searching.

For the granule level the descriptor file contains only a list of the attributes and a specification of how values will be assigned to them. This information is used to generate a Metadata Configuration File (MCF), which is delivered to the Data Processing Subsystem or the Ingest Subsystem on demand. The descriptor also carries valid values and ranges for Product-Specific attributes and a list of services for the collection. See Section J.3 for roles and responsibilities for preparation of the collection and granule metadata.

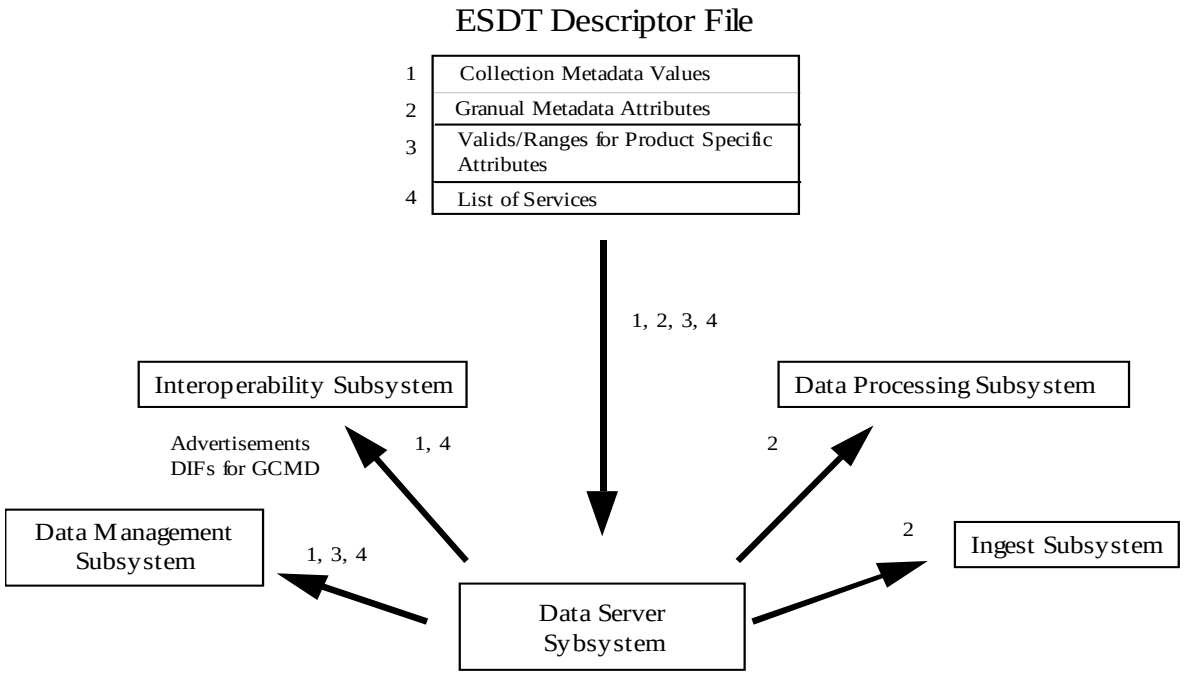


Figure J-1. Metadata Flow in ECS

ECS uses collection metadata in the descriptor file to advertise a new collection and to update the system-wide data dictionary. From the granule attributes in the ESDT descriptor Science Data Server produces a Metadata Configuration File (MCF) that is filled in during product generation (for products produced within ECS) or filled in during ingest processing (for external data delivered to ECS).

Data providers and producers should exercise special care when selecting granule attributes to represent their data and in writing values for those metadata. An error in a collection attribute or value can be corrected by manual edits to the ESDT descriptor file but an error in a granule attribute or value can affect all members of the collection in the inventory.

J.2.3 Mandatory Metadata

In 420-TP-016-003 ("*Backus-Naur Format (BNF) Representation of the B.0 Earth Science Data Model for the ECS Project*) designates the minimum set of metadata attributes that must be supplied for different categories of product managed by ECS. The categories of metadata support are as follows:

Full level of metadata - required for products generated with ECS

Intermediate level of metadata - required for products generated outside ECS, but ingested and used within ECS

Limited level of metadata - applies to all other data sets.

The selection of metadata attributes for inclusion in any given product is done at the time the ESDT descriptor for that product is built. The toolkit can check that granule-level mandatory attributes have been populated during granule production, as described in Section J.6.2.

J.3 Procedures and Support

An MCF file is necessary for each output produced by a PGE that is to be stored on the Science Data Server. If multiple granules with the same ESDT are being produced, the same MCF is reused for each granule.

In prior SDP Toolkit versions, an all-inclusive MCF template was included and the science software developer had to edit the template to customize it to the particular need. Since the structure of each MCF is tightly couple to the definition of corresponding ESDT, it was deemed necessary to **substantially change** t**his process for science software development for ECS Release B.0.**

EOSDIS metadata support staff are available to assist with generation of both ESDT descriptor files and MCFs to be used in science algorithm testing. If the name of an ECS contact for metadata and ESDTs has not been provided to you, please send an email message requesting such support to landover_PGSTLKIT@raytheon.com. Specific questions regarding metadata or ESDTs may also be sent to this email address.

J.3.1 Generating the Metadata Configuration File in Release B.0

Beginning with B.0, ECS provided you with a Metadata Configuration File built from the tool MetaData Works. The MetaData Works is no longer active, therefore, the steps in utilizing MetaData Works to build a new MCF file has been deleted in this section.

Beginning with 7.20 ECS delivery the system will also accept XML metadata files from the SIPS.

J.4 The Granule Metadata Population Process

Figure J-5 is a schematic of the process by which data granules and their metadata are generated. In Step 1 Science Data Server Science notifies Data Processing of the arrival of input data needed to produce new data granules. When all the inputs are available, Science Data Processing then requests Science Data Server to return a Metadata Configuration File (MCF) that is to be filled in with values for the granule metadata attributes (Step (2)). In Step (3) Science Data Processing generates new data granules (i.e., a science data product) by running a Product Generation Executive (PGE) together with a Process Control File that defines the input and output file locations and other control parameters to the PGE. In Step (4) the PGE, using the SDP Toolkit, writes values for the granule metadata attributes into the MCF. These steps are described in detail in Sections J.5 and J.6 of this Appendix.

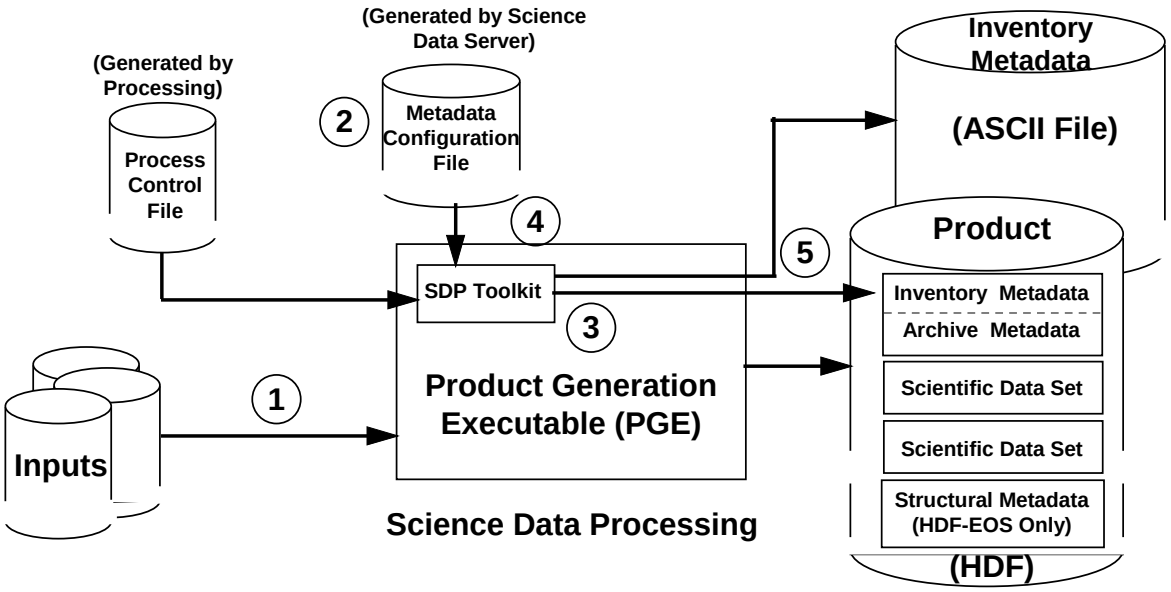


Figure J-2. Science Data Production and Archival Scenario.

In Step (5) the populated MCF (inventory metadata) is written into both the data product (if it is in HDF) and to an ASCII metadata file which is then subsequently inserted into Science Data Server to populate the inventory database tables.

Information describing the internal structure of an HDF-EOS data product, and its data elements, is attached to the granule by the PGE using HDF-EOS calling sequences. This "structural" metadata is not used to populate the inventory, rather it is used to support the services which may be performed upon the granule. There is no direct association between the metadata groups set up in the MCF and the structural metadata. Note that there is no need to define structural metadata within an MCF. The structural metadata is automatically generated by the HDF-EOS APIs and has the attribute name "structmetadata.N" (N=0...9). This is described in more detail in the HDF-EOS Users Guide.

J.5 Metadata Toolkit Usage

Section 6 of the main body of the Toolkit Users' Guide gives the calling sequences for each metadata toolkit functions, along with examples of code in both ANSI C and FORTRAN. The purpose of this section is to explain how the tools work together and provide a step-by-step example.

J.5.1 Overview

Multiple MCFs may be opened and written to from within a single PGE. There are four metadata tools that are used in conjunction with an MCF that must be called in a specific sequence for each MCF. First, each MCF must be initialized with **PGS_MET_Init.** Each call to

PGE_MET_Init returns a unique identifier for that MCF. Values generated within the PGE are assigned to attributes in the MCF using **PGS_MET_SetAttr**, which is called once per attribute. After all values have been assigned, **PGS_MET_Write** is used to write the metadata to the product as well as a separate ASCII metadata file. Finally, **PGS_MET_Remove** frees up memory occupied by the MCFs. Before a call to **PGS_MET_Write** it is required that the HDF file, into which the metadata is going to be written, is opened. For HDF files of HDF4 type one can use HDF4's **SDstart** (**sfstart** for FORTRAN) to open HDF file and obtain a SD ID to pass into **PGS_MET_Write**. However, for opening a HDF file of HDF5 type and obtaining SD ID one needs to call **PGS_MET_SDstart** (**pgs_met_sfstart** for FORTRAN). The function **PGS_MET_SDstart** (**pgs_met_sfstart** for FORTRAN) can also be used for opening HDF files of HDF4 type. The HDF files opened by a call to **PGS_MET_SDstart** (**pgs_met_sfstart** for FORTRAN) must be closed by a call to **PGS_MET_SDend** (**pgs_met_sfend** for FORTRAN) after writing metadata.

Three additional metadata tools are used from within the PGE to read in metadata values. **PGS_MET_GetSetAttr** returns the value of any metadata attribute in an MCF that has loaded into memory. Two other tools may be called independently of any MCF: **PGS_MET_GetPCAttr** returns the value of metadata attributes from input files (either embedded metadata in HDF-EOS files, or independent ASCII metadata files), and **PGS_MET_GetConfigData** returns the value of runtime metadata from the Process Control File.

J.5.2 Example

This example includes retrieval of metadata from an HDF file and from the PCF, and setting and writing attributes in a new product. These code fragments are in C. Consult Section 6 for the equivalent calls in FORTRAN. Some concepts introduced in this example are explained in further detail in Section J.6.

First a value for the runtime parameter with the name "Runtime_ID" is read from the user-defined runtime parameters section of the Process Control File using PGS_MET_GetConfigData:

```
/* get values from PCF */
ret =
PGS_MET_GetConfigData("Runtime_ID",&rtid)
```

Next, PGS_MET_GetPCAttr is used to read a value for the attribute EquatorCrossingLongitude from the inventory metadata block of an HDF input file whose fileID is 10265. Another call to PGS_MET_GetPCAttr reads in a value MAX_DELTA from a separate ASCII file with fileID 5731. (See notes under PGS_MET_GetPCAttr in Section 6.2.1.4 concerning specification of metadata input files in the PCF.)

```
/* get value from metadata block of input file */
ret =
PGS_MET_GetPCAttr(10265,1,INVENTORYMETADATA,"EquatorCrossingLongitude",&val);

/* get value from ASCII metadata file */
ret =
```

```
PGS_MET_GetPCAttr(5731,1,INVENTORYMETADATA,"MAX_DELTA",&dval);
```

Then PGS_MET_Init is used to read into memory an MCF whose fileID is 10250 and check its syntax. An array mdHandles is returned with pointers for each metadata block in the MCF (see Sections 6.2.1.4 and J.6.1 for details).

```
/* Initialize an MCF into memory */
ret =
PGS_MET_Init(10250,mdHandles);
```

The PGE now calculates a new value for LocalVersionID writes it to the MCF held in memory. PGS_MET_SetAttr locates the attribute name and assigns a value to it.

```
/* assign value to attribute in MCF */
ret =
PGS_MET_SetAttr(mdHandles[1],"LocalVersionID",&val);
```

A value already assigned to the MCF in memory is needed by the PGE so PGS_MET_GetSetAttr is used to retrieve it.

```
/* Read back in value of attribute in memory */
ret =
PGS_MET_GetSetAttr(mdHandles[1],"SensorCharacteristicValue.1”,value)
```

The PGE has finished setting all the values which are mandatory in the MCF, but there is still some relevant granule information the data provider wishes to add. The PGE accomplishes this by writing this information to the product specific metadata group in the INVENTORYMETADATA section of the MCF. A suffix “1” is added to the second argument of the call to distinguish multiple uses of these parameters, as discussed in Section J.6.

```
/* assign value to Product-Specific Attribute */
ret =
PGS_MET_SetAttr(handles[1],"AdditionalAttributeName.1",”Max_Slope”);

ret =
PGS_MET_SetAttr(handles[1],"ParameterValue.1",”57.5”)
```

The PGE now writes some granule metadata to the archive block of the MCF. This metadata will not be searchable in the inventory database tables, but it will be readable using toolkit calls.

```
/* assign value to attribute in MCF in Archive block*/
ret =
PGS_MET_SetAttr(handles[2],"Runtime_ID",&rtid);
```

Once the algorithm has finished retrieving and setting values in the memory, the final stage is to write the inventory and archive metadata blocks to the product. PGS_MET_Write writes the metadata blocks to an HDF (HDF4 or HDF5 type) file as HDF global attributes (an unfortunate duplication of terms; an HDF attribute should not be confused with an individual metadata

attribute). Note that a separate call to PGS_MET_Write is required for the inventory and archive metadata blocks.

```
/* Write Metadata Blocks to HDF output file */
ret =
PGS_MET_Write(mdHandles[1],"coremetadata",sdid1);

ret =
PGS_MET_Write(mdHandles[2],"archivemetadata",sdid2);

/* Write all Metadata Blocks to ASCII output file */
ret =
PGS_MET_Write(mdHandles[0],NULL,101);

/* Remove MCF from memory*/
ret =
PGS_MET_Remove()
```

It is imperative that PGS_MET_Write be called in order to generate an ASCII metadata output file, as this is the means by which inventory database tables are populated during Insert of the product into the Data Server Subsystem. This ASCII metadata output file is generated automatically when the INVENTORYMETADATA section is written to an HDF product. If a non-HDF output product is being generated that will be archived by ECS, it is necessary to use PGS_MET_Write to generate this ASCII metadata output file using a variation in the calling sequence. The user must give the mdHandle[0], reserved to point to the whole MCF, the second arguments as NULL, and the final argument as the file ID. In either case the metadata output file is given the same name as the data product output file, but with the suffix ".met" attached. If the file ID in PGS_MET_Write is set to NULL, a default ASCII dump file is created. More examples of writing metadata to product files are given in the HDF-EOS Users' Guide, Volume 1, Section 8.

The format of the metadata written into the product or output as a separate ASCII file is Object Description Language, ODL, which is described in more detail in the next section.

J.6 Structure of the Metadata Configuration File (MCF)

As described in Section J.3, the MCF is the vehicle for populating granule-level metadata attributes which are then attached to product granules and used to populate the inventory database tables. Since the MCF is a byproduct of the ESDT descriptor file, it should not be necessary for data producers to be cognizant of its structure and syntax. However, this section of the Appendix is being provided to assist anyone having a need to create or modify an MCF. Another reason for being familiar with the format of the MCF is that the populated MCF, which is written to the product file and passed as an ASCII file to Science Data Server, is in Object Description Language (ODL) and is nearly identical in format to the MCF that serves as input to the PGE.

The structure of the MCF allows users to distinguish between two types of metadata: that which will be used to populate the inventory in the data server and therefore will be available for searching on granules, and that which is important to the description of the granule and therefore

needs to be kept with the granule as it is archived, but need not be searchable. These separate parts (or Mastergroups as they are called in the MCF) are called Inventory and Archive metadata.

J.6.1 MASTERGROUPS

The MCF consists of one or more "master groups." The only required MASTERGROUP is called INVENTORYMETADATA which contains the metadata attributes whose values will be inserted into the inventory database tables, as well as being written to (or exported with) the product. Any number of additional MASTERGROUPs can be created and values can be written to them, but these metadata values will not appear in the inventory database and will only written to the product. Each MASTERGROUP is written as an HDF global attribute using PGS_MET_Write. Inventory metadata must be written to an HDF global attribute named "coremetadata." By convention, there is just one additional MASTERGROUP named ARCHIVEMETADATA and it is written to an HDF global attribute named "archivemetadata."

It should be noted that the PGS_MET_Write tools will automatically create multiple HDF global attributes, e.g. coremetadata.1, coremetadata.2, coremetadata.3, ... to accommodate a MASTERGROUP that exceeds the HDF size limits for global attributes. When this HDF file is used as input to another PGE, the multiple global attributes are recognized by the toolkit as a single block. However, other HDF tools may need to be instructed to access the attributes individually.

The MCF must start with:

```
GROUP = INVENTORYMETADATA
      GROUPTYPE = MASTERGROUP
```

and end that master group with:

```
END_GROUP = INVENTORYMETADATA
```

If additional, non-inventory metadata are to be included in the MCF, they must appear between:

```
GROUP = ARCHIVEDMETADATA
      GROUPTYPE = MASTERGROUP
```

and:

```
END_GROUP = ARCHIVEDMETADATA
```

A parameter called GROUPTYPE is assigned the value MASTERGROUP to signal the toolkit that all attributes enclosed within the named group are to be treated as a block. This distinguishes the mastergroups from other groupings of attributes as described below.

J.6.2 MCF Hierarchy

The hierarchical organization of attributes in the MCF follows as closely as possible the conceptual model of ECS metadata as described in DID-311. The MCF is written in Object Description Language, or ODL, which enables a hierarchical organization of information using Groups, Objects, and Parameters. Groups are used to represent Classes in the ECS Data Model and Objects are used to represent individual metadata attributes. Each Object is described by a number of Parameters. The following example will be used in describing each of these terms:

```
GROUP = ECSDataGranule

      OBJECT = SizeMBECSDataGranule
            Data_Location = "DSS"
            NUM_VAL = 1
            TYPE = "DOUBLE"
            Mandatory = "FALSE"
      END_OBJECT = SizeMBECSDataGranule


      OBJECT = DayNightFlag
            Data_Location = "PGE"
            NUM_VAL = 1
            TYPE = "STRING"
            Mandatory = "TRUE"
      END_OBJECT = DayNightFlag

      OBJECT = ProductionDateTime
            Data_Location = "TK"
            NUM_VAL = 1
            TYPE = "DATETIME"
            Mandatory = "TRUE"
      END_OBJECT = ProductionDateTime

      OBJECT = LocalVersionID
            Data_Location = "PGE"
            NUM_VAL = 1
            TYPE = "STRING"
            Mandatory = "TRUE"
      END_OBJECT = LocalVersionID

END_GROUP = ECSDataGranule
```

In this example the Group ECSDataGranule consists of four objects, SizeMBECSDataGranule, DayNightFlag, ProductionDateTime, and LocalVersionID. Each object is described using four Parameters: Data_Location, NUM_VAL, TYPE, and Mandatory. These four parameters are required for every object in the MCF (except objects which are containers as described below).

In the MCF an object can be described using the parameters: Data_Location, Mandatory, NUM_VAL, TYPE, CLASS and Value. All parameter names are case insensitive and their arguments (i.e. what appears to the right of the "=" sign) must be in quotes, unless the argument is numeric. A description of each parameter follows.

Data_Location - The metadata tools are used to set metadata values for a product granule coming from three possible input sources—the Metadata Configuration File itself, the Process

Control File and the PGE. The parameter Data_Location indicates the source of population. Data_Location must be set for every object.

"MCF" - When the Data_Location is equal to "MCF" the object will have its value set in the MCF using the "Value = " parameter. This option is used for attributes whose values will remain the same for all granules. An example is the mandatory attribute collection ShortName, which is included in each granule for identification purposes.

"PGE" - When the Data_Location is equal to "PGE" the object will have its value set by the science software using the PGS_MET_SetAttr metadata tool. This is the way most objects are set.

"PCF" - The Process Control File contains all file input and output specifications as well as runtime parameters. When the Data_Location is equal to "PCF" the object will have its value set automatically during initialization of the MCF when using PGS_MET_Init. The Toolkit will locate the Object name within the USER DEFINED RUNTIME PARAMETERS section of the PCF and the corresponding value will be assigned to the Object. The attribute name to be searched on must be written between the first and second delimiters in the PCF, and its corresponding value between the second and third delimiters . (For further details on the format of the PCF, see Appendix C of this document.) For example, if the PCF contained:

```
10255|PLATFORMSHORTNAME|"TRMM"
```

then

```
ret = PGS_MET_GetConfigData("PLATFORMSHORTNAME",&val)
```

would return "TRMM" in val. In the PCF quotes are only necessary when the datatype of the value in the MCF is STRING. If an attribute is to be stored in the PCF as a runtime parameter, the attribute name must be in UPPER case and must appear only once in the PCF.

"NONE" - used only in conjunction with container objects as discussed below.

The MCF may also provide place holders for metadata attributes that will be set at a later stage in a granule's life. Other possible values for Data_Location include:

- "DAAC" for attributes that will be given values later at the DAACs, (e.g. OperationalQualityFlag),
- "DP" for attributes that will be given values later by the Data Producer, (e.g. ScienceQualityFlag),
- "DSS" for attributes that will be given values later by the Data Server Subsystem, (e.g. SizeMBECSGranule), and
- "TK" for attributes automatically given values by the Toolkit, (e.g. ProductionDateTime.

Mandatory - This parameter, which can have the values “TRUE” or “FALSE,” provides a means for checking the metadata population process. PGS_MET_Write returns an error if no value has been set for an attribute which has Mandatory = “TRUE”. If no value has been set for a attribute which has Mandatory = "FALSE" a warning will be returned. In the former case PGS_MET_Write sets the value to “NOT_SET”. Any attempt to insert a data granule into Data Server will fail if any of the attributes have Mandatory=“TRUE” but an attribute value of “NOT_SET.” An attribute with Mandatory = "FALSE" that is not set will be omitted from the output metadata file.

Attributes designated in the ECS Data Model as being mandatory should have the mandatory flag set to “TRUE”. Science Data Server may reject any granule that is lacking mandatory metadata.

Type - The type parameter allows the metadata tools to set the correct datatype for attributes written by the PGE. The permitted values for this parameter are: “DATE”, “TIME”, "DATETIME", "INTEGER", "DOUBLE", "STRING" and "UNSIGNEDINT. DATETIME is of the form 1997-04-03T12:36:00”.

Note that since ODL does not support unsigned integers, the value written by the PGS_MET_Write tool may appear negative, but the Toolkit handles any conversion between signed and unsigned values based on the TYPE. Users must remember that setting of datatype they require will be using ODL specific types. This does not interfere with the users own setting datatype of values returned from the Toolkit call (e.g. a float may be converted to a double).

NUM_VAL - An attribute can be single-valued or a one-dimensional array of values. NUM_VAL gives the maximum number of elements in an attribute value array. Any number of values up to this limit may be set. If NUM_VAL is greater than one and the value is set in the PCF or the MCF, the array is enclosed in parentheses: e.g. (“value1”,”value2”,...) or (12, 34, 45, 88). To set a array of values using the metadata tools, PGS_MET_SetAttr is called once with an array as the attribute value. See notes for PGE_MET_SetAttr in Section 6.2.1.4 which describe conventions for partial filling of arrays.

Value - This parameter is only present in the MCF template when the Data_Location = “MCF”. In the output metadata file, after the metadata population is complete, the parameter Value appears for all attributes. As noted previously, if a value has not been filled by either the PGE, PCF or MCF, then either a default value will be set, or the attribute will not be written, and an error or warning will be returned from PGS_MET_Write..

CLASS - In the ECS Data Model some classes may be repeated multiple times. For example, in a granule the attribute SensorCharacteristic may be used once to describe a sensor’s operating temperature and again to give a reference voltage. The CLASS parameter is used to signal the toolkit than the attribute named by an object in the MCF will be written to multiple times and that each write should create a separate instance of that object in the metadata output file. This is discussed in the next section.

J.6.3 Setting Multiple Attribute Values

Some attribute names can be used multiple times. The permitted multiplicity is specified in the ECS Data Model (see 420-TP-016-003). To allow an attribute or group of attributes to be multiply defined they must be bounded by an object called a “container.” This object container is then bounded by an affiliated group name. The CLASS for the container object must be set to "M", where M stands for multiple. For example:

```
GROUP = SensorCharacteristic
      OBJECT = SensorCharacteristicContainer
      Data_Location = "NONE"
      Class = "M"
      Mandatory = "TRUE"

            OBJECT = SensorShortName
                  Data_Location = "PGE"
                  Mandatory = "TRUE"
                  Class = "M"
                  TYPE = "STRING"
                  NUM_VAL = 1
            END_OBJECT = SensorShortName

            OBJECT = SensorCharacteristicName
                  Data_Location = "PGE"
                  Mandatory = "TRUE"
                  Class = "M"
                  TYPE = "STRING"
                  NUM_VAL = 1
            END_OBJECT = SensorCharacteristicName

            OBJECT = SensorCharacteristicValue
                  Data_Location = "PGE"
                  Mandatory = "TRUE"
                  Class = "M"
                  TYPE = "STRING"
                  NUM_VAL = 1
            END_OBJECT = SensorCharacteristicValue

      END_OBJECT = SensorCharacteristicContainer
END_GROUP = SensorCharacteristic
```

To use an attribute multiple times the PGS_MET_SetAttr tool must be called with a CLASS suffix to the attribute name. For example, using CLASS = 1:

```
PGS_MET_SetAttr(mdHandles[1],"SensorShortName.1”,”SHIRS”)

PGS_MET_SetAttr(mdHandles[1],"SensorCharacteristicName.1”,”CentralWavelength”)

PGS_MET_SetAttr(mdHandles[1],"SensorCharacteristicValue.1”,”450.1”)
```

The actual suffix used is not important but integer increments are advised. CLASS is only present for objects and groups which have multiple instances. Collection-level metadata

attributes are used to define a data type for this and other “self-defining” attributes (see Section 6.4).

A new instance of the container object is created by the tools on output each time attribute is used. For example, if a second sensor characteristic were set using:

```
PGS_MET_SetAttr(mdHandles[1],"SensorShortName.2”,”AVHRR”)

PGS_MET_SetAttr(mdHandles[1],"SensorCharacteristicName.2”,”Model_No”)

PGS_MET_SetAttr(mdHandles[1],"SensorCharacteristicValue.1”,”AH773Z”)
```

Note that SensorCharacterisiticValue is numeric in the first case and alphanumeric in the second case. Although the same attribute in the MCF is being used multiple times, its type is set only once. Therefore, in the MCF its type must be “string” and the values being assigned must be set in quotes inside PGS_MET_SetAttr. The true datatype for sensor characteristic (or any of the self-defining attributes) is set in the collection-level metadata. The value of the attribute SensorCharacterisitcDataType would anyone someone to convert the string returned for SensorCharacterisitcValue to it’s correct data type.

The metadata output file would look like this:

```
GROUP = SensorCharacteristic
      OBJECT = SensorCharacteristicContainer
      CLASS = "1"

            OBJECT = SensorShortName
                  CLASS = "1"
                  NUM_VAL = 1
                  VALUE = “AVHRR”
            END_OBJECT = SensorShortName

            OBJECT = SensorCharacteristicName
                  CLASS = "1"
                  NUM_VAL = 1
                  VALUE = “Central Wavelength”
            END_OBJECT = SensorCharacteristicName

            OBJECT = SensorCharacteristicValue
                  CLASS = "1"
                  NUM_VAL = 1
                  VALUE = “450.1”
            END_OBJECT = SensorCharacteristicValue

      END_OBJECT = SensorCharacteristicContainer

      OBJECT = SensorCharacteristicContainer
      CLASS = "2"

            OBJECT = SensorShortName
                  CLASS = "2"
                  NUM_VAL = 1
                  VALUE = “AVHRR”
            END_OBJECT = SensorShortName
```

```
            OBJECT = SensorCharacteristicName
                  CLASS = "2"
                  NUM_VAL = 1
                  VALUE = “Model_No”
            END_OBJECT = SensorCharacteristicName

            OBJECT = SensorCharacteristicValue
                  CLASS = "2"
                  NUM_VAL = 1
                  VALUE = “AH773Z”
            END_OBJECT = SensorCharacteristicValue

      END_OBJECT = SensorCharacteristicContainer
END_GROUP = SensorCharacteristic
```

This example shows the ODL structure of the metadata written to the product, and what parameters are kept to describe the objects. Not all parameters held within the MCF are written to the metadata output file. The parameters which are written for each object are: NUM_VAL, CLASS and the VALUE associated with the object.

Data_Location must be consistent for all objects within a container. In other words, you cannot have the Data_Location for ExclusionGRingFlag be “MCF” and then have GRingPointLatitude with Data_Location = “PGE” within the same GPolygonContainer.

J.6.4 Product-Specific Attributes

The ECS Data Model contains a number of the attributes that are termed self describing. These are used to extend the ECS Data Model by allowing the definition of new attributes. Since these are usually defined solely for a particular product, they are sometimes referred to as Product-Specific Attributes or PSAs. The classes holding attributes in this category are: AdditionalAttributes and SensorCharacteristics. The classes VerticalSpatialDomain and RegularPeriodic can also be considered self-describing.

Self-describing attributes are defined by classes which include a name, datatype, description and value for the new attribute. The name, datatype and description are defined at the collection level, while the value is given at the granule level (i.e. written to the granule’s metadata using the toolkit) along with the attribute name so that the association with the collection-level attributes can be made. Self-describing groups can be set multiple times by a PGE and the product-specific attribute value can be a single-dimensional array by setting NUM_VAL greater than 1. The AdditionalAttributes class has the following construction in an MCF (see example of previous section as well):

```
GROUP = AdditionalAttributes

OBJECT = AdditionalAttributesContainer
```

```
        Data_Location = "NONE"
        Class = "M"
        Mandatory = "TRUE"

        /*  AdditionalAttributes  */
        OBJECT = AdditionalAttributeName
              Data_Location = "PGE"
              Mandatory = "TRUE"
              TYPE = "STRING"
              Class = "M"
              NUM_VAL = 5
        END_OBJECT = AdditionalAttributeName

        /*  InformationContent  */
        GROUP = InformationContent

              Class = "M"

              OBJECT = ParameterValue
                    Data_Location = "PGE"
                    Mandatory = "TRUE"
                    TYPE = "STRING"
                    NUM_VAL = 5
              END_OBJECT = ParameterValue

        END_GROUP = InformationContent

END_OBJECT = AdditionalAttributesContainer
END_GROUP = AdditionalAttributes
```

In the example above, NUM_VAL is the largest number of possible values (i.e. the largest possible array size) of any attributes that will be set using "AdditionalAttributes." For example, if two product-specific attributes will be set, one single-valued and the second an array of dimension 5, then NUM_VAL must be set to 5.

Note that although PSAs are written as name/value pairs, they are read in the same fashion as core attributes. That is, PGS_MET_SetAttr is called twice to write out a PSA, once to populate AdditionalAttriubteName, then once to set ParameterValue. However, PGS_MET_GetSetAttr or PGS_MET_GetPCAttr need only be called once, with the value given to AdditionalAttributeName in order to obtain the value given to ParameterValue.

J.7 Metadata in HDF vs. non-HDF input and Output Files

Once populated, the MCF carries the granule-level metadata information. This information is delivered to Science Data Server to populate the inventory database tables. In order for the data product to be most useful, this information needs to be either embedded within the product or closely tied to it. If the output product is in HDF, the toolkit automatically writes the granule-level metadata to the product as one or more HDF Global Attributes. HDF attributes have a 64K size limit, so the toolkit automatically generates additional attributes to hold all metadata being written.

If the output product is not in HDF a separate ASCII metadata file must be generated. This is accomplished using PGS_MET_Write is the manner described in main body of the Toolkit documentation.

J.8 MCF Syntax

The MCF is closely based on Object Description Language (ODL) libraries. Most information pertinent to PGE developers about ODL and its functionality is contained within this document. Additional information is available at the WWW address http://pds.jpl.nasa.gov/stdref/chap12.htm. ODL is based on a parameter = value syntax.

- ODL handles parameters and values in Upper case. The metadata toolkit converts all character strings in the MCF to upper case upon initialization into memory.
- ODL only recognizes a character string value when it is in quotation marks.
- ODL accepts only UTC Time/Date which must be in CCSDS ASCII format (A or B)
- ODL will only accept INTEGER, UNSIGNEDINT, DOUBLE , DATETIME or STRING as a value for type

This page intentionally left blank.

Appendix K. POSIX Systems Calls Usage Policy

This appendix outlines the usage policy for the set of POSIX system API calls as outlined in:

> *IEEE Std 1003.1: POSIX Part 1: System Application Program Interface (API) [C Language]*
>
> *IEEE Std 1003.9: POSIX FORTRAN77 Language Interfaces, Part 1: Binding for System Application Program Interface [API]*

In general, the policy attempts to guard access to routines that may impact the SDPS where system resource management is an issue. This will be accomplished by restricting access to the standard POSIX system calls, as described below. The complete set of routines is listed in the "Identifier Index" of *IEEE Std 1003.1*, and in the body of IEEE Std 1003.9.

Table C–1 provides general policy "guidelines" for various classes of system routines. These guidelines are then used in determining the appropriate disposition for each of the POSIX system call functions on an individual basis. The general policy guidelines include:

- **Toolkit**—The described functionality is either accessible to the user via a "shadowing" routine in the PGS Toolkit, or it is used internally by the Toolkit itself. The Toolkit routine may be a simple subroutine call (or macro) wrapper around the "shadowed" function, or it may provide additional functionality that may be useful to the SDPS in accomplishing its resource management objectives. Direct calls to the respective POSIX API calls are prohibited within science algorithm code.
- **Prohibited**—Access to the described functionality is prohibited. Direct calls to the respective POSIX API calls are prohibited within science algorithm code.
- **Allowed**—Access to the described functionality is allowed through the standard POSIX API calls. The Toolkit itself makes calls to these routines in addition to those listed in the Toolkit category.

The algorithm integration and test facility will include "code checkers" to screen science algorithms for adherence. These code checkers will be provided as part of the PGS Toolkit to support the development of policy compliant algorithms. This should greatly facilitate the algorithm integration and test procedure.

Table K-1. POSIX Call Guidelines By Class

Class	Description	Policy Guideline
Process control	Process creation and termination; interprocess signaling and synchronization	Toolkit
Memory	Memory allocation, deallocation, and mapping	Toolkit
File I/O	File I/O routines; directory manipulation routines	Toolkit
Stream I/O	Stream I/O routines	Toolkit
Error / environment	Error recording and reporting; environment access	Toolkit
Ownership	Process ownership and groups; file ownership, permissions, and creation/access times	Prohibited
Miscellaneous	Math, "is...", "str...", and time functions	Allowed
Terminal I/O	Terminal I/O and characteristics	Prohibited
Status	System and resource status (read only)	Allowed

Tables K–2 through K–10 constitute a listing of the entire set of POSIX C API calls, organized by class and policy as described above. Table K–11 provides a listing of the FORTRAN77 specific language library calls that do not have C API counterparts. Entries in **bold** indicate that a Toolkit "shadow" function has been created to perform this functionality.

Table K-2. POSIX Calls: Process Control

Toolkit Routines	Prohibited Routines	Allowed Routines
exec...() **_exit()** fork() **sig...()** sleep() wait() waitpid()	abort() alarm(), PXFALARM() exec(), PXFEXEC...() _exit(), PXFEXIT() PXFFASTEXIT() PXFFORK() kill(), PXFKILL() pause(), PXFPAUSE() PXFSIG...() PXFSLEEP() PXFWAIT() PXFWAITPID() ftp find nice rlogin	exit()

Table K-3. POSIX Calls: Memory

Toolkit Routines	Prohibited Routines	Allowed Routines
calloc() **free()** **malloc()** **realloc()**		calloc() free() malloc() realloc()

Table K-4. POSIX Calls: File I/O

Toolkit Routines	Prohibited Routines	Allowed Routines
access() close() creat() dup() dup2() **lseek()** open() pipe() read() remove() rename() **tmpfile()** **tmpnam()** write()	access(), PXFACCESS() fclose(), PXFCLOSE() creat(), PXFCREAT() dup(), PXFDUP() dup2(), PXFDUP2() PXFLSEEK() PXFOPEN() PXFPIPE() PXFREAD() cd cp rcp read chdir(), PXFCHDIR() PXFRENAME() closedir(), PXFCLOSEDIR() fpathconf(), PXFFPATHCONF() getcwd(), PXFGETCWD() PXFWRITE() link(), PXFLINK() mkdir(), PXFMKDIR() mkfifo(), PXFMKFIFO() opendir(), PXFOPENDIR() pathconf(), PXFPATHCONF() readdir(), PXFREADDIR() rewinddir(), PXFREWINDDIR() rmdir(), PXFRMDIR() unlink(), PXFUNLINK() utime(), PFXUTIME() PXFUMASK(), PXFUNAME()	

Table K-5. POSIX Calls: Stream I/O

Toolkit Routines	Prohibited Routines	Allowed Routines
fclose() fcntl(), fdopen() fileno() **fopen()** freopen()	setbuf() stdin stdout stderr PXFFCNTL() PXFFDOPEN() PXFFILENO() clearerr() PXFPOSIXIO() at,atq,atrm	feof() ferror() fflush(), PXFFLUSH() fgetc(), PXFFGETC() fgets() fprintf() fputc(), PXFFPUTC() fputs() fread() fscanf() fseek(), PXFFSEEK() ftell(), PXFFTELL() fwrite() getc(), PXFGETC() putc(), PXFPUTC() sprintf() sscanf() ungetc()

Table K-6. POSIX Calls: Error/environment

Toolkit Routines	Prohibited Routines	Allowed Routines
	assert() atexit()	assert(), getenv(), PXFGETENV() perror() IPXFARGC() PXCLEARENV() PXFGETARG() PXFSETENV()

Table K-7. POSIX Calls: Ownership

Toolkit Routines	Prohibited Routines	Allowed Routines
getpid() getppid()	PXFGETPID() PXFGETPPID() chgrp mkdir ln chmod(), PXFCHMOD() chown(), PXFCHOWN() getegid(), PXFGETEGID() geteuid(), PXFGETEUID() getgid(), PXFGETGID() getgrgid(), PXFGETGRGID() getgrnam(), PXFGETGRNAM() getgroups(), PXFGETGROUPS() getlogin(), PXFGETLOGIN() getpgrp(), PXFGETPGRP() getpwnam(), PXFGETPWNAM() getpwuid(), PXFGETPWUID() getuid(), PXFGETUID() setgid(), PXFSETGID() setpgid(), PXFSETPGID() setsid(), PXFSETSID() setuid(), PXFSETUID() umask(), PXFUMASK() utime(), PXFUTIME()	

Table K-8. POSIX Calls: Miscellaneous

Toolkit Routines	Prohibited Routines	Allowed Routines
l	localeconv() setlocale()	localeconv() setlocale()

Table K-9. POSIX Calls: Terminal I/O

Toolkit Routines	Prohibited Routines	Allowed Routines
	cfgetispeed(), PXF...() cfgetospeed(), PXF...() cfsetispeed(), PXF...() cfsetospeed(), PXF...() ctermid(), PXFCTERMID() getchar() gets() isatty(), PXFISATTY() lp, lpr, lpstat mail printf() putchar() puts() scanf() tc...(), PXFTC...() ttyname(), PXFTTYNAME()	

Table K-10. POSIX Calls: Status

Toolkit Routines	Prohibited Routines	Allowed Routines
fstat(), PXFFSTAT() stat(), PXFSTAT() uname(), PXFUNAME() PXFIS...()	PXFSTAT()	sysconf(), PXFSYSCONF() times(), PXFTIMES()

Table K-11. POSIX Calls: FORTRAN77 Language Library

Toolkit Routines	Prohibited Routines	Allowed Routines
open() close() read(5,…) read(*,…) write(6,…) write(*,…)	READ*... READ(*,...) WRITE(*,...) WRITE(6,...)	PXFCALLSUBHANDLE() IPXFCONST() PXFCONST() PXFGETSUBHANDLE() PXFISCONST() PXFSTRUCTCOPY() PXFSTRUCTCREATE() PXFSTRUCTFREE() PXFSUBHANDLE()PXF<TYPE> GET() PXF<TYPE>SET() PXFA<TYPE>GET() PXFA<TYPE>SET() PXFE<TYPE>GET() PXFE<TYPE>SET()

Appendix L. Ephemeris and Attitude File Formats

EOSDIS Spacecraft Ephemeris and Attitude Data Specification: Contents and Structure

A Requirements Document for Incoming Data for the SDP Toolkit

Version 3, Including TRMM, Terra, Aqua, and Aura Specific Items

Peter D. Noerdlinger and Guru Tej S. Khalsa

SM & A Corporation

and

Robert Kummerer

Raytheon Systems Corporation

May 2004

This document specifies the form for incoming spacecraft ephemeris and attitude data for the EOSDIS Science Data Processing Toolkit. Sample file structures below show the required data format, and the required and optional data. This October 2003 version also explains some of the functionality of DPREP, the generic name of spacecraft-specific software that transforms incoming spacecraft data to Toolkit form. Although the file format is generic, there are platform-specific items that are normally included. There is also a permanent change to the reference frame for the attitude rates, for all spacecraft after TRMM. Version 1 of this document named the Euler angles in the order peculiar to TRMM; Version 2 denotes them generically as three angles, to be in the order that is specified in the header. Thus, for TRMM our names were yaw, pitch, and roll for the 3-2-1 Euler angle order. For Terra and later spacecraft the order will be simply Euler angle 1, Euler angle 2, Euler angle 3. (For Terra these are yaw, roll, and pitch, i.e. 3,1,2). This setup is more flexible and can accommodate orders that repeat, such as 3,1,3, Euler's original choice[1]. *Tables 2I-S describe an important addition to the file metadata: for both ephemeris and attitude, if range or continuity checking have been done, the metadata must include the limits or thresholds used to set the error flags. Accompanying documentation (the DPREP specifications) outlines the algorithm used for this kind of quality checking.*

This document is written in the context that original ephemeris and attitude data will be processed into files suitable for the SDP Toolkit[2]. In that processing, checks may be performed,

[1] For further explanation, see, e.g. Spacecraft Attitude Determination and Control, Ed. J. R. Wertz (D. Reidel, Dordrecht, 1985), especially Appendix E.

[2] Documents Terra Spacecraft Ephemeris and Attitude Data Processing (document 500-EMD-001), Aqua Spacecraft Ephemeris and Attitude Data Processing (document 500-EMD-002), and Aura Spacecraft Ephemeris and Attitude Data Processing (document 500-EMD-003) describe ephemeris and attitude preprocessing for the Terra, Aqua, and Aura platforms.

small gaps may be filled, the units and even the reference frame may be changed, etc. The Toolkit itself has interpolation capability; the term "data repair" is used for any filling of data gaps in processing *before* the Toolkit, and the result is a "repaired" point. The omission of any item in this document does not vitiate requirements specified elsewhere (e.g. metadata requirements).

The size space to be allocated for UR in the file headers is variable, in order to accommodate changing PDPS requirements.

Some data items must be present and must be in a specified format and units. Others (such as identification fields) are required, but the format is not set in this document. Some items or groups of items (in particular, the orbital elements) are optional. Certain data must eventually be cast into ODL form and supplied to the archive system as inventory metadata. For EOS program spacecraft this process will be done within the EOSDIS system; for other spacecraft a decision will have to be taken as to how this part is done, but we flag the required fields with underlines in any case.

The ODL formats are specified in the following document:

http://pds.jpl.nasa.gov/stdref/Chapter12.pdf

The SDP Toolkit has many functions that can be helpful in translating foreign data formats into Toolkit standard form. Toolkit staff will be glad to work with outside data providers to facilitate translations using these tools, which include time translations and reference frame changes.

*1. **Definitions and Preliminaries***

1.1 Files and File Structures

Because ephemeris and attitude data may arrive separately, and at different intervals, the ephemeris and attitude data must be kept in separate files. In EOSDIS the data are generally kept in HDF files, with metadata assigned a separate section at the end. The Toolkit reads flat binary data files, which may also be used as temporary or permanent vehicles for storage of ephemeris and attitude data. In this document, no distinction will be made between the two kinds of file, but metadata will be distinguished from data. Conceptually, metadata are used to identify the contents of a file, such as the spacecraft identification, the time span, etc. The metadata are segregated to facilitate data base access. *File headers are by definition classified as metadata, for they are used to identify files for retrieval.* This definition is notwithstanding any specification as to where metadata physically lie. I.e., if, for example, HDF standards put metadata at the end of the file physically, then the physical end is the "Header." The distinction between metadata and data is only relevant to the SDPS archiving system; either kind can be written and read as normal C data.

1.2 Time Standards

All ASCII times shall be UTC and be conformant with CCSDS Format A standards as explained in the CCSDS Blue Book (CCSDS Blue Book, Issue 2, *Time Code Formats*; CCSDS 301.0–B–2) issued by the Consultative Committee for Space Data Systems (NASA Code OS, NASA, Washington DC 20546, April 1990) and the EOSDIS SDP Toolkit User's Guide (SDP Toolkit

Users Guide for the ECS Project, which is on line on the *World Wide Web* at http://edhs1.gsfc.nasa.gov/database/ECSCatalog.html). The date must be included. All binary times shall be in Toolkit Internal Time. Toolkit Internal Time, secTAI93, is defined as continuous seconds from UTC midnight, Jan 1, 1993. Normally kept as a double precision (64 bit) number, it suffices to maintain microsecond resolution from the late 1970's to beyond the year 2020. Functions in the Toolkit readily translate between Toolkit internal time, spacecraft clock time, UTC, GPS, and other popular time streams. Users are advised to use Toolkit or other reliable software, which includes leap seconds, to obtain this time. Some UNIX and C time conversion utilities omit leap seconds when calculating time intervals, a serious error.

1.3 Units and Reference Coordinate Systems for the Ephemeris and the Orbital Elements

Position and velocity data must be in SI units (m and m/s), angles in radians, and angular rates in radians per second. The ephemeris shall be in J2000. The orbital elements, if provided, can be in J2000, TOD, or TOR (see below); a required field identifies which system was used.

1.4 Orbital Elements

The metadata for each file of orbital data may contain orbital elements; if these are unavailable, the relevant fields can be left unpopulated. The osculating Keplerian elements are chosen, generally consistent with the approach in the following document (see note after the table for the exception):

Goddard Trajectory Determination System (GTDS) Mathematical Theory, Revision 1 Edited by A.C. Long et al, Goddard Space Flight Center Code 550, Document FDD/552-89/001 or CSC/TR-89/6001, 1989.

Note that the epoch of the elements can be different from that of the reference frame wherein they are defined. The epoch of the reference frame must be must be shown in Table 3[3]. Note that the orbital elements will often be defined in the native coordinate system, while the ephemeris is required to be in J2000.

Table 1. Keplerian Orbital Elements (1 of 2)

Symbol	Meaning
keplerElements[0]	Semi-major axis of spacecraft orbit (m)
keplerElements[1]	Orbital eccentricity
keplerElements[2]	Inclination (radians)

[3] Note that in Goddard Space Flight Center Code 500 Standard Ephemeris Data Product, for some time, the right ascension of the ascending node and the argument of perigee are in the reverse order from ours. All other GSFC FDD data products follow the order shown in Table 1. Our early data preparation segment will put the data in the order shown in Table 1. See the Goddard Space Flight Center Flight Dynamics Division Interface Control Document for Generic Data Product Formats, Document 553-FDD-91/028 (GSFC 1991).

Table 1. Keplerian Orbital Elements (2 of 2)

Symbol	Meaning
keplerElements[3]	Right ascension of the ascending node (radians)[1]
keplerElements[4]	Argument of the perigee (radians)[1]
keplerElements[5]	Mean anomaly at epoch (radians)
keplerEpochTAI	Epoch of the elements (SI sec from 1993-01-01T00Z)

1.5 Identification of Other Frames

While orbital elements are not essential to SDPS processing, they are provided for herein both for checking purposes and so as to preserve incoming data that are often present. Although the orbital ephemeris is to be in J2000, the orbital elements could be defined in True of Date (TOD) ECI or True of Reference (TOR) ECI, in J2000 ECI or in B1950 ECI. We will require a tag in the metadata, such as "J2000" or "TOR" showing the reference system of the elements.

1.5.1 Fixed Epoch Inertial Systems - J2000 ECI and B1950:

The B1950 and J2000 reference systems are defined in the Astronomical Almanac and the GTDS document mentioned in Section 1.4. Their axes are along fixed directions in inertial space. The AM and PM Series of spacecraft will have their *original* ephemerides defined in J2000; *ephemerides from other platforms may have to be transformed (e.g. by DPREP).* Toolkit functions are available to assist with this work.

1.5.2 Inertial Systems at Other Epochs - TOD and TOR

True of Date (TOD) means the inertial system obtained by precessing and nutating J2000 to the current time of the orbital data, and True of Reference (TOR) means the inertial system obtained by precessing and nutating J2000 to some other epoch, generally the time of the start of the first orbit in the data file. Even in a day, the change in the coordinate axes due to the change in precession and nutation is generally < 0.3 arc seconds per axis, equivalent to < 15 meters in position for the a low Earth orbit spacecraft (total for 3 axes). Nevertheless, for completeness, in the case of TOR, the epoch should be provided.

1.6 Orbit Numbers

The orbit numbers will represent full orbits from the beginning of the mission. Each orbit after the first begins with an upward (ascending) equator crossing. The crossing will be determined in the same coordinate system as the native data (Section 1.5 above). The orbit up to the first ascending node is orbit number 1 for TRMM, orbit 0 for Terra, TBD for later spacecraft. The Terra orbit number is established by FDD and the DPREP value will be forced into agreement by operator action if necessary.

1.7 Longitude of Equator Crossing

The terrestrial longitude of the crossing of the Earth's equator on an orbit is to be identified in the metadata, to facilitate later retrieval of swath data. The downward equator crossing longitude and time of crossing, *in True of Date coordinates,* are to be determined and placed in the metadata. Note that there is no conflict in tagging the orbits with data from the downward crossing, although the orbit began at the upward; the upward crossing will be near the middle of each orbit. This actually may avoid confusion between the longitude of the equator crossing in one orbit and the next. The crossing must be defined in true-of-date or true-of-epoch, where the epoch is within a day of the actual date. Use of the downward crossing will optimize the association of orbits with daylight swaths for the Terra spacecraft. *If the science data granules arriving at EOSDIS from the spacecraft will not contain or be processed into any swath data (i.e. all the data will be scene data that fit in bounding rectangles of limited extent, or the data will all be global data sets) then the longitudes and times of equator crossing need not be populated unless required elsewhere in EOSDIS requirements.*

1.8 Actual versus Commanded Variables for Attitude Data; Attitude Rate Differences

Data providers should be aware that incoming spacecraft data are sometimes in the form of differences from commanded quantities, especially for attitude. In that case, the commanded and the difference quantity must be summed before transmittal to the Toolkit. In the case of Terra, the interface documentation for the ancillary data (to which ECS is not a party) states that the attitude and attitude rates (prior to DPREP processing) are in orbital coordinates, but verbal and e-mail statements from the Terra office have stated that the attitude and rates are relative to commanded. Furthermore, the commanded data are not provided in the ancillary data. Because of these problems, users are advised to employ FDD attitude, which is obtained for EOSDIS from the housekeeping, and not the ancillary data, by the GSFC Flight Dynamics Division. This attitude is absolute in orbital coordinates and the rates are the projection of the absolute (J2000) inertial angular velocity on the spacecraft coordinate axes. The meaning of the attitude in the *ancillary* data, although it is processed by DPREP as well, is not guaranteed. It is supposed, however, that the values and rates will most likely be defined in orbital coordinates, except during maneuvers. Thus the mean pitch rate in the ancillary data will probably be zero, while in the FDD data it is (outside of maneuvers) very close to the negative of the instantaneous orbital angular rate, any small difference being due to variations of the attitude from nominal.

1.9 Reference System for the Attitude

The reference system for the attitude will in all cases be geocentric orbital coordinates. The **y** axis is the instantaneous negative orbit normal, the **z** axis is toward Earth center and the **x** axis is along the cross product of the **y** unit vector with the **z** unit vector.

1.10 Specification of the Attitude

The attitude will be specified in terms of Euler angles, and the angular rates about the three principal spacecraft axes. Any additional attitude parameters (such as changes in "flying mode", or flags showing that maneuvers are in progress) must be absorbed into either the Euler angles or the quality flags. For example, if in "flying forward" at zero roll and pitch, the yaw is zero, then "flying backwards" can be defined as yawAngle = π.

1.11 Meaning of the Euler Angles and Rates

The Euler angles will always be ordered *within the records* according to the actual Euler Angle Order. Each angle will be in radians, and will be defined positive when the rotation is in the sense of a right handed screw along its positive axis - i.e., the right handed rule is applied when looking outwards from the nominal spacecraft center. The ranges of the Euler angles are not restricted; the usual ranges are given in Spacecraft Attitude Determination and Control, Ed. J. R. Wertz (D. Reidel, Dordrecht, 1985), pp. 763 - 764. For TRMM, the rates will be instantaneous rates of rotation about the three body axes **x**, **y**, and **z**, defined positive in the same right-hand sense. Thus, for example, if the spacecraft is flying Eastward and "backwards", with its body x axis along the negative velocity, a positive roll rate will mean that the North surface or appendages are descending, the South ascending. Note that body axis rates are not, in general, the same as rates of change of the Euler angles. When the angles are all small, and no axis is repeated, the rates of rotation about the body axes are approximately equal to the Euler angle rates, but the order is always (roll, pitch, yaw) and is *not* adjusted to match the Euler angle order. *Thus, it will not be unusual that the order of the Euler angle values will not match that of the rates*. For TRMM, the pitch rate is to be "stripped" in that it is relative to orbital and not to inertial reference axes. For Terra and later EOSDIS spacecraft, the rates will the projection of the absolute (J2000) inertial angular velocity on the spacecraft coordinate axes. See http://newsroom.gsfc.nasa.gov/sdptoolkit/faq.html#q16.

For a description of the transformation of the rates between orbital and inertial frame, see http://newsroom.gsfc.nasa.gov/sdptoolkit/FinalRateAtt.html.

1.12 Order of the Euler Angles

The file metadata must provide the Euler Angle Order; i.e. a mapping of **x** = roll, **y** = pitch, **z** = yaw into the set 1,2,3. The order is to represent rotations that the spacecraft would undergo in achieving its actual attitude starting from alignment with orbital coordinates. For example, if the spacecraft must be put through a pitch, then a yaw, then a roll to achieve its true attitude starting from perfect alignment with the orbital system, the Euler Angle Order is 2,3,1.

1.13 Quality Flags

Quality flags definitions for ephemeris and attitude are outlined here. In actual cases, the flags bits are set according to spacecraft-specific criteria that should be explained and supported with references to original documents. Table 2A shows the usage of the platform generic quality flags. Tables 2B-H show the usage of platform-specific quality flags for TRMM, Terra, Aqua, and Aura, respectively. In a specific case, not all fields may be populated. For the latter tables, the usage could be quite different, for different spacecraft, but bit 16 is reserved for a platform-specific flag, *if* the data provider intends to send data packets considered to be quite unreliable. (The alternative is to send no data in such cases).

The SDP Toolkit tools for ephemeris and attitude access are user-callable, but are also used by higher-level tools. The user interface differs somewhat in the two cases. When the access tool is called directly, it passes the flags on to the user. In the other case, for example if the user is accessing geolocation services, the interface has to be different, because the user cannot access the flags *per se* through other tools which call the ephemeris access tools. In early Toolkit

releases, an error was returned only when large data gaps existed; the flags were ignored. The current and future Toolkits implement fuller recognition of quality flags by higher-level tools that call the ephemeris tool. Thus, both missing data and bad quality data can result in warning or error messages.

The Toolkit now implements, as default behavior, data rejection when bit 16 is set. The SDP Toolkit function PGS_EPH_ManageMasks() enables the user to set a quality flag mask, if desired, in the Process Control File, enforcing rejection based on other bits of her or his own choice. For this reason, data providers are encouraged to establish practical definitions of flag bits suitable for users to check questionable points. In particular, bits 2, 5, 6 and 9, if set, can be used by users to reject points. These represent the large gap and red variation limits. It is generally supposed that some range or continuity checks have been imposed on the data, and will be reflected in some of the flags ("yellow" and "red" limits exceeded). Because the checks could be range or continuity checks, accompanying documentation should explain the procedure, i.e. the meaning of these limits. Such documentation is available for TRMM, Terra, Aqua, and Aura.

So that the parameters used in checking will be available in the data sets themselves, we are requesting that the parameter values be listed along the line shown forTRMM, Terra, Aqua, and Aura, for example, in Tables 2I-Q. We are planning that the red-limit bit be set so as to statistically reject not more than 0.01% of the data when the variation is normal statistics, and the yellow-limit bit be set so as to reject not more than 0.1%. The actual decision will be made in each case by ESDIS. Note that for Terra L0 ephemeris, ESDIS has directed that DPREP shall replace points outside the red or yellow limits by a quartic least squares fit, when this results in the replacement of data segments whose length is shorter than or equal to 1 minute of time. When this is done, the data repair bit is set and any bits that were set to indicate the existence of a problem (limit exceeded, etc.) will be unset, for the following reason: Most of the time users will access the ephemeris data via other tools. In that case, the only means available to select bit patterns for rejection is the use of the tool PGS_EPH_ManageMasks(). That tool allows a simple mask comparison test, not complicated logic such as would be required to accept repaired data with another "trouble" bit set, but reject bad data that could not be repaired because of the gap length. When defective data segments longer than the maximum 60-second gap length exist, entire *replacement data sets* will be obtained from FDD. For Terra, these data sets will have packet time interval 1.0s, rather than the 1.024s in L0 data. This will be documented in the ephemeris header as shown in Table 3.

DPREP also provides a summary of the quality checks, in the form of quality assurance statistics, as shown in Table 2R-S.

Table 2A. Platform-Generic Quality Flags (1 of 2)

Bit	Bit Assignment	Description
0	Overall Quality Summary	Set if any quality check is failed; unset for ideal data. Data point can still be useful even if this bit is set; scrutiny of the other bits would be required however. Bits 1 and 16 are unset in this instance of ideal data.
1	Data State Summary	Set if any generic data quality bit is set (*bits 2 - 11*)
2	Red Limit Low Exceeded[4]	Low red limit has been exceeded.
3	Yellow Limit Low Exceeded	Low yellow limit has been exceeded.
4	Yellow Limit High Exceeded	High yellow limit has been exceeded.
5	Red Limit High Exceeded	High red limit has been exceeded.
6	Long Data Gap Follows[5]	A significant data gap originally followed this data point.
7	Short Data Gap Follows	A minor data gap originally followed this data point.
8	Short Data Gap Precedes	A minor data gap originally preceded this data point.
9	Long Data Gap Precedes	A significant data gap originally preceded this data point.
10	Point is a repaired data point[6]	Used for points inserted by software *prior to Toolkit* (interpolated).
11	Quality flag problem	Quality data not available (bits 0-5 not meaningful)[7].

[4] The red and yellow limits are typically limits for the variation from nominal or commanded or deviations of a single point from a local fit to the data. We recommend setting the yellow limit so that for the kinds of error expected about one point per thousand, but no more, will be flagged yellow. We recommend setting the red limit such that not more than one point in ten thousand would normally be flagged. The thresholds for the red and yellow limits are platform specific. Accompanying documentation would explain the details. The limits can be different for orbit and for attitude data. Red limits, if defined, should be chosen carefully as future Toolkit modifications might cause rejection of points with bits 2 or 5 set. When a red limit is exceeded, the yellow is obviously exceeded also, so when bit 2 is set, bit 3 should be set, and when bit 5 is set, bit 4 should be set. When a flag represents several items (such as both position and velocity, or all 3 Euler angles) it is set for the worst of them.

[5] The number of points constituting a tolerable gap is platform specific. Accompanying documentation should show what size gaps are flagged. The gaps may or may not have been filled by interpolation. Filled points are indicated in bit 10. The definitions of “long” and “short” gaps can be different for orbit and for attitude data.

[6] A “repaired” point has been interpolated after original data processing - typically to fill a data transmission gap.

Table 2A. Platform-Generic Quality Flags (2 of 2)

Bit	Bit Assignment	Description
12	No data available	SDP Toolkit unable to find data at the requested timestamp.
13	Unassigned	Reserved for SDP Toolkit use.
14	Interpolated data point	SDP Toolkit interpolation performed in deriving data point.
15	Unassigned	Reserved for SDP Toolkit use.

Note: Bits 1-15 are Platform Generic Flags are for general data quality flagging, and are intended to apply to all platforms. Bits 12-15 are reserved for SDP Toolkit use. Bit 0 is *least significant*.

Table 2B. TRMM Platform-Specific Quality Flags (1 of 2)

Bit	Bit Assignment	Description
16	Platform-Specific Flag	Set if any platform-specific quality bit is set[8].
17	QAC Flag	Data transmission flagged in QAC list.
18	Yaw Acquisition	Set if ACS yaw acquisition in progress.
19	Yaw Maneuver	Set if ACS yaw maneuver in progress.
20	Yaw Update Inaccurate	Set if ACS has yet to check current yaw. Error in yaw attitude up to 0.5 degrees anticipated.
21	Contingency Mode Flag	Set if ACS is operating in a degraded state due to an Earth sensor failure.
22	Inertial Hold Flag	Spacecraft is flying in inertial space locked mode.
23	Earth Acquisition	Set if ACS Earth acquisition in progress.
24	Yaw Update Indeterminate	Set while ACS yaw determination completes following a delta-V maneuver. No error in yaw attitude expected, but can be suspect.
25	Delta-V Maneuver	Set if delta-V maneuver in progress.

---

[7] For example, if the quality check involves testing smoothness, isolated points (with gaps on each side) cannot be checked.

[8] In the case of TRMM, only bits 17 and 18 would be fatal for use of the data. For other platforms, the preparer needs to decide on and document the bit patterns.

Table 2B. TRMM Platform-Specific Quality Flags (2 of 2)

Bit	Bit Assignment	Description
26	Flying +X Forward	Set if flying with +X axis in the forward direction.
27	Flying –X Forward	Set if flying with -X axis in the forward direction.
28	Flying –Y Forward	Set if flying with -Y axis in the forward direction.
29-31	Unassigned	Available for other platform-specific data, quality or other.

Note: Bits 17 through 31 are Platform-Specific Flags reserved for data flagging except that bit 16 is common to all platforms. Bit 31 is *most significant*. The definitions outlined here are for TRMM.

Table 2C. Terra Platform-Specific Quality Flags

Bit	Bit Assignment	Description
16	Platform-Specific Flag	Set if any platform-specific quality bit is set[9].
17	Safe Mode Flag	Spacecraft has initiated Spacecraft Safe Mode; data are unusable.
18-31	Unassigned	Available for other platform-specific data, quality or other.

Note: Bits 17 through 31 are Platform-Specific Flags reserved for data flagging except that bit 16 is common to all platforms. Bit 31 is *most significant*. The definitions outlined here are for Terra.

Table 2D. Aqua Platform-Specific Quality Flags (1 of 2)

Bit	Bit Assignment	Description
16	Platform-Specific Flag	Set if any platform-specific quality bit is set[10].
17	Bad Status Word	Attitude remains unprocessed due to invalid attitude system mode in Status Word 2 data stream.
18	Missing Status Word	Attitude remains unprocessed due to missing attitude system mode in Status Word 2 data stream.

[9] In the case of Terra, bit 17 is fatal for use of the data. Bit 16 is also set when bit 17 is set.

[10] In the case of Aqua, bits 17, 18, and 19 are fatal for use of the data. Bit 16 is also set when any of bits 17-19 are set.

Table 2D. Aqua Platform-Specific Quality Flags (2 of 2)

Bit	Bit Assignment	Description
19	Bad Ephemeris Data	Attitude remains unprocessed due to poor-quality or missing ephemeris data.
20-31	Unassigned	Available for other platform-specific data, quality or other.

Note: Bits 17 through 31 are Platform-Specific Flags reserved for data flagging except that bit 16 is common to all platforms. Bit 31 is *most significant*. The definitions outlined here are for Aqua.

Table 2E. Aura Platform-Specific Quality Flags

Bit	Bit Assignment	Description
16	Platform-Specific Flag	Set if any platform-specific quality bit is set[11].
17	Bad Status Word	Attitude remains unprocessed due to invalid attitude system mode in Status Word 2 data stream.
18	Missing Status Word	Attitude remains unprocessed due to missing attitude system mode in Status Word 2 data stream.
19	Bad Ephemeris Data	Attitude remains unprocessed due to poor-quality or missing ephemeris data.
20-22	Operating Mode	GN&C operating mode from Status Word 2. Table 2F describes operating mode values.[12]
23-25	Operating Mode Transition Flag.	Operating mode transition flag. Tables 2G-H describe the operating mode transition flag.
26-31	Unassigned	Available for other platform-specific data, quality or other.

Note: Bits 17 through 31 are Platform-Specific Flags reserved for data flagging except that bit 16 is common to all platforms. Bit 31 is *most significant*. The definitions outlined here are for Aura.

[11] In the case of Aura, bits 17, 18, and 19 are fatal for use of the data. Bit 16 is also set when any of bits 17-25 are set.

[12] Refer to Aura Spacecraft Ephemeris and Attitude Data Processing (document 500-EMD-003) for more information on the operating mode and mode transition flag.

Table 2F. GN&C Operating Mode Description

Binary	Decimal	GN&C Operating Mode Description	Science Data Possible?
000	0	Mode Zero	No
001	1	Attitude Hold	Yes
010	2	Sun Hold	No
011	3	Fine Point	Yes
100	4	Earth Point	Yes[13]
101	5	Sun Point	No

Table 2G. Operating Mode Transitions and Mode Transition Values

	To	0	1	2	3	4	5
From	**0**	0	7	7	7	7	7
	1	7	0	7	1	2	7
	2	7	7	0	7	7	7
	3	7	3	7	0	4	7
	4	7	5	7	6	0	7
	5	7	7	7	7	7	0

[13] MLS may be able to produce science data during earth point mode so it is treated as a science-producing mode in this operation.

Table 2H. Interpretation of Mode Transition Values (1 of 2)

Binary	Decimal	Interpretation
000	0	No transition has occurred between the Status Word 2 records that bracket this attitude.
001	1	A transition from *attitude hold* to *fine point*. The spacecraft is cycling from propulsion mode to normal science mode. Instruments may be able to take data during both these modes.
010	2	A transition from *attitude hold* to *earth point*. The spacecraft is cycling from propulsion mode to either a stand-by mode or a safe mode. It is possible that MLS can take data during *earth point* mode.
011	3	A transition from *fine point* to *attitude hold*. The spacecraft is cycling from *fine point* to propulsion for orbit adjustment. Instruments can take data during both these modes.
100	4	A transition from *fine point* to *earth point*. MLS may be able to take data during *earth point* mode.
101	5	A transition from *earth point* to *attitude hold*. This transition is not likely.
110	6	A transition from *earth point* to *fine point*.
111	7	Any transition between, into, or out of non-science data-taking modes. Some of these transitions are not possible, e.g. it is not possible to go from *mode zero* to *fine point* mode. In general, a value of 7 in this field will signal that the data may be unusable if DPREP is able to process it at all.

Table 2I. Quality Checking Parameters – TRMM Platform-Specific (1 of 2)
EDOS-Supplied Attitude and FDD-Supplied Ephemeris

Symbol	Meaning
qaParameters[0]	Number of records required for populating quality check queue
qaParameters[1]	Short gap interval in seconds
qaParameters[2]	Long gap interval in seconds
qaParameters[3]	Absolute position error red low limit
qaParameters[4]	Absolute position error yellow low limit

Table 2I. Quality Checking Parameters – TRMM Platform-Specific (2 of 2)

Symbol	Meaning
qaParameters[5]	Absolute position error yellow high limit
qaParameters[6]	Absolute position error red high limit
qaParameters[7]	Position error change yellow limit
qaParameters[8]	Position error change red limit
qaParameters[9]	Position error standard deviation yellow limit
qaParameters[10]	Position error standard deviation red limit
qaParameters[11-15]	Unused

Table 2J. Ephemeris Quality Checking Parameters – Terra Platform-Specific (1 of 2)

EDOS-Supplied Ephemeris

Symbol	Meaning
qaParameters[0]	Maximum ephemeris quality check window size in number of data points
qaParameters[1]	Minimum ephemeris quality check window size in number of data points
qaParameters[2]	Long gap size in seconds
qaParameters[3]	Position vector yellow limit in standard deviations or meters
qaParameters[4]	Position vector red limit in standard deviations or meters
qaParameters[5]	Velocity vector yellow limit in standard deviations or meters per second
qaParameters[6]	Velocity vector red limit in standard deviations or meters per second
qaParameters[7]	Absolute or standard deviations limit check method flag
qaParameters[8]	Unused
qaParameters[9]	Absolute position vector maximum in meters

Table 2J. Ephemeris Quality Checking Parameters – Terra Platform-Specific (2 of 2)

Symbol	Meaning
qaParameters[10]	Absolute position vector minimum in meters
qaParameters[11]	Absolute velocity vector maximum in meters per second
qaParameters[12]	Absolute velocity vector minimum in meters per second
qaParameters[13-15]	Unused

Table 2K. Ephemeris Quality Checking Parameters – Terra Platform-SpecificFDD-Supplied Ephemeris (replacement data)

Symbol	Meaning
qaParameters[0]	Long gap size in seconds
qaParameters[1]	Absolute position vector maximum in meters
qaParameters[2]	Absolute position vector minimum in meters
qaParameters[3]	Absolute velocity vector maximum in meters per second
qaParameters[4]	Absolute velocity vector minimum in meters per second
qaParameters[5-15]	Unused

Table 2L. Ephemeris Quality Checking Parameters – Aqua Platform-Specific FDD-Supplied Ephemeris

Symbol	Meaning
qaParameters[0]	Long gap size in seconds
qaParameters[1]	Absolute position vector maximum in meters
qaParameters[2]	Absolute position vector minimum in meters
qaParameters[3]	Absolute velocity vector maximum in meters per second
qaParameters[4]	Absolute velocity vector minimum in meters per second
qaParameters[5-15]	Unused

Table 2M. Ephemeris Quality Checking Parameters – Aura Platform-Specific FDD-Supplied Ephemeris

Symbol	Meaning
qaParameters[0]	Long gap size in seconds
qaParameters[1]	Absolute position vector maximum in meters
qaParameters[2]	Absolute position vector minimum in meters
qaParameters[3]	Absolute velocity vector maximum in meters per second
qaParameters[4]	Absolute velocity vector minimum in meters per second
qaParameters[5-15]	Unused

Table 2N. Attitude Quality Checking Parameters – Terra Platform-Specific EDOS-Supplied Attitude[14]

Symbol	Meaning
qaParameters[0]	Long gap size in seconds
qaParameters[1-15]	Unused

Table 2O. Attitude Quality Checking Parameters – Terra Platform-Specific FDD-Supplied Attitude[15] *(1 of 2)*

Symbol	Meaning
qaParameters[0]	Long gap size in seconds
qaParameters[1]	Absolute roll angle maximum in radians
qaParameters[2]	Absolute pitch angle maximum in radians

[14] The L0 (EDOS) attitude data are not checked for range limits, or continuity (spikes) because of conflicting information as to the reference system for the attitude and rates. The available written agreements, to which ECS is not a party, state that the attitude is relative to orbital coordinates, but personnel involved in the production of these data have stated that the attitude is relative to the commanded attitude, which is not present in the EDOS/L0 data stream.

[15] The FDD attitude data are not checked for range limits, or continuity (spikes) because such data are deemed to be without error as delivered from FDD. Range checking is performed and any violation results in the prompting for a replacement dataset from FDD.

Table 2O. Attitude Quality Checking Parameters – Terra Platform-Specific FDD-Supplied Attitude[16] (2 of 2)

Symbol	Meaning
qaParameters[3]	Absolute yaw angle maximum in radians
qaParameters[4]	Absolute roll angle minimum in radians
qaParameters[5]	Absolute pitch angle minimum in radians
qaParameters[6]	Absolute yaw angle minimum in radians
qaParameters[7]	Absolute angle rate maximum in radians per second
qaParameters[8]	Absolute angle rate minimum in radians per second
qaParameters[9]	Absolute roll angle yellow limit in radians
qaParameters[10]	Absolute pitch angle yellow limit in radians
qaParameters[11]	Absolute yaw angle yellow limit in radians
qaParameters[12]	Absolute roll angle red limit in radians
qaParameters[13]	Absolute pitch angle red limit in radians
qaParameters[14]	Absolute yaw angle red limit in radians
qaParameters[15]	Absolute x angle rate yellow limit in radians per second
psQaParameters[0]	Absolute y angle rate yellow limit in radians per second
psQaParameters[1]	Absolute z angle rate yellow limit in radians per second
psQaParameters[2]	Absolute x angle rate red limit in radians per second
psQaParameters[3]	Absolute y angle rate red limit in radians per second
psQaParameters[4]	Absolute z angle rate red limit in radians per second
psQaParameters[5-15]	Unused

[16] The FDD attitude data are not checked for range limits, or continuity (spikes) because such data are deemed to be without error as delivered from FDD. Range checking is performed and any violation results in the prompting for a replacement dataset from FDD.

Table 2P. Attitude Quality Checking Parameters – Aqua Platform-Specific EMOS-Supplied Attitude[17]

Symbol	Meaning
qaParameters[0]	Attitude data long gap size in seconds
qaParameters[1]	Absolute roll angle maximum in radians
qaParameters[2]	Absolute pitch angle maximum in radians
qaParameters[3]	Absolute yaw angle maximum in radians
qaParameters[4]	Absolute roll angle minimum in radians
qaParameters[5]	Absolute pitch angle minimum in radians
qaParameters[6]	Absolute yaw angle minimum in radians
qaParameters[7]	Absolute angle rate maximum in radians per second
qaParameters[8]	Absolute angle rate minimum in radians per second
qaParameters[9]	Status Word 2 long gap size in seconds
qaParameters[10-15]	Unused

Table 2Q. Attitude Quality Checking Parameters – Aura Platform-Specific EMOS-Supplied Attitude[13] (1 of 2)

Symbol	Meaning
qaParameters[0]	Attitude data long gap size in seconds
qaParameters[1]	Absolute roll angle maximum in radians
qaParameters[2]	Absolute pitch angle maximum in radians
qaParameters[3]	Absolute yaw angle maximum in radians
qaParameters[4]	Absolute roll angle minimum in radians
qaParameters[5]	Absolute pitch angle minimum in radians
qaParameters[6]	Absolute yaw angle minimum in radians

[17] The EMOS attitude data are not checked for range limits, or continuity (spikes) because such data are deemed to be without error as delivered from EMOS. Range checking is performed and any violation results in the prompting for a replacement dataset from EMOS.

Table 2Q. Attitude Quality Checking Parameters – Aura Platform-Specific EMOS-Supplied Attitude (2 of 2)

Symbol	Meaning
qaParameters[7]	Absolute angle rate maximum in radians per second
qaParameters[8]	Absolute angle rate minimum in radians per second
qaParameters[9]	Status Word 2 long gap size in seconds
qaParameters[10-15]	Unused

Table 2R. Platform Generic Quality Assurance Statistics

Symbol	Meaning
qaStatistics[0]	QA Percent Interpolated Data
qaStatistics[1]	QA Percent Missing Data
qaStatistics[2]	QA Percent Out-of-Bounds Data
qaStatistics[3]	Unused

Table 2S. Aqua and Aura Platform-Specific Quality Assurance Statistics

Symbol	Meaning
psQaStatistics[0]	QA Percent Missing Status Words
psQaStatistics[1]	QA Percent Bad Status Words
psQaStatistics[2]	QA Percent Bad Ephemeris Data
psQaStatistics[3]	Unused

1.14 Versions

All the data within one incoming file must be the same version. If an original provider supplies, for example, files beginning with definitive data and ending with predicted, the parts must be segregated; the Toolkit does not deal with different versions, though the process control system will allow file substitutions.

1.15 Cautions

In the handling of incoming data, especially from historic data sets, and data sets foreign to the EOSDIS fleet of spacecraft, it is important to remember that in practice the units for position and velocity and the order of the Euler angles within the packets might be different from our specification. The angles are also likely to be in different units. Furthermore, many spacecraft using horizon sensors are referenced and even controlled to geodetic nadir. Euler angles referenced in this way must be transformed to geocentric orbital coordinates before they are acceptable, as will be done for TRMM. It is also possible that for some historic or foreign data sets the Euler angles and their order as originally produced may represent an alias and not an alibi transformation (*See Malcolm D. Shuster, A Survey of Attitude Representations in J. Astronaut. Sci.* ***41****, 439 - 517 (1993). As explained on pp. 494-495, the attitude matrix for alibi transformations is the transpose of that for alias*.)

2. Summary of Data and Metadata Structures

The tables in this section summarize the various structures, first for ephemeris, then for attitude. The order is (1) file header structure, (2) UR List, (3) record structure, and (4) metadata structure (ephemeris only). The structures are shown in tables, but contain the necessary punctuation, preamble and termination to constitute C++ structures. Some systems pad structures with extra bits. We have defined structural elements in such a way that, on several machines familiar to us, the size of the structure as stored is equal to the sum of the sizes of its listed component elements. We write and read the structures as structures. Our tables do not include machine-dependent padding bits that may be present on other machines, causing the structure to have length greater than the sum of its listed components.

Table 3. Ephemeris Header Standard Structure (Metadata) (1 of 2)

(One per file)

(Underlined items must propagate to inventory metadata)

typedef struct

{

// Type	Name	Meaning
Char	spacecraftID[24];	// Spacecraft Name
Char	asciiTimeRange[48];	// Start stop times to nearest hour or better, in ASCII[18]
Char	source[32];	// Source of the data[19]

[18] Example: “1999-04-11T06Z to 1999-04-12T06Z”. This partial redundancy with the double precision start and stop times is for readability.

Table 3. Ephemeris Header Standard Structure (Metadata) (2 of 2)

// Type	Name	Meaning
Char	version[8];	// Version number (default = 1)
PGSt_double	startTime;	// Ephemeris dataset start time, secTAI93
PGSt_double	endTime;	// Ephemeris dataset end time, secTAI93
PGSt_real	interval;	// Expected interval between records, SI seconds[20]
PGSt_uinteger	nURs;	// Number of input dataset universal references
PGSt_uinteger	nRecords;	// Number of ephemeris records
PGSt_uinteger	nOrbits;	// Number of orbits spanned, including fragments
PGSt_uinteger	orbitNumberStart;	//Number of first orbit or part orbit in file
PGSt_uinteger	orbitNumberEnd;	//Number of last orbit or part orbit in file
Char	keplerRefFrame[8];	// Reference Frame: e.g. "TOD", "TOR" or "J2000" of the Keplerian elements[21]
PGSt_double	keplerElements[6];	// Osculating Keplerian elements at epoch[17]
PGSt_double	keplerEpochTAI;	// TAI 93 Epoch of the Reference Frame[17,22]
PGSt_real	qaParameters[16];	// Ephemeris data quality processing parameters
PGSt_real	qaStatistics[4];	// Quality assurance statistics
Char	spare[216];	// Pad to 512 bytes

} PGSt_ephemHeader;

---

[19] This field might read: “GSFC FDD”, or “TONS”. Terms like “filtered”, “smoothed”, or “unfiltered” are allowable.

[20] This is the normal interval as in the original data stream, not accounting for data gaps, clock error, etc.

[21] These three fields are not required to be populated - but if one is, all three should be.

[22] *If the reference frame is B1950 or J2000, this field can be unpopulated. If it is "TOD" or "TOR" the value should be supplied, since the last Keplerian element itself is the time at which they osculate to the orbit, while the reference frame may be defined at a somewhat different time*. Note that this reference frame applies only to the elements of orbit only; the actual ephemeris is converted to J2000 by DPREP. The elements give only a thumbnail view of the orbit, so the actual ephemeris data should be used when accuracy is required.

Table 4. Ephemeris Record Standard Structure

(One per record)

typedef struct
{

// Type	Name	Meaning
PGSt_double	secTAI93;	// Date and time as seconds from 1-1-93, secTAI93
PGSt_double	position[3];	// X component of position vector, meters
PGSt_double	velocity[3];	// X component of velocity vector, meters/sec
PGSt_uinteger	qualityFlag;	// Ephemeris data quality flag.
Char	spare[4];	// Pad structure to 64 bytes

} PGSt_ephemRecord;

Table 5. Ephemeris Orbit Metadata Standard Structure

(One per orbit)

(Underlined items must propagate to inventory metadata; orbitAscendTime to archive metadata)

typedef struct
{

// Type	Name	Meaning
PGSt_uinteger	orbitNumber;	// Orbit number, from beginning of mission
Char	spare[4];	// Pad previous element to 8 bytes
PGSt_double	orbitAscendTime;	// Time of upward TOD equator crossing, secTAI93
PGSt_double	orbitDescendTime;	// Time of downward TOD equator crossing, secTAI93
PGSt_double	orbitDescendLongitude;	// Orbit down-crossing terrestrial longitude, radians

} PGSt_ephemMetadata;

Table 6. Attitude Header Standard Structure (Metadata)

(One per file)

(Underlined items must propagate to inventory metadata)

typedef struct
{

// Type	Name	Meaning
char	spacecraftID[24];	// Spacecraft Name
char	asciiTimeRange[48];	// Start and stop times to nearest hour or better, in ASCII[11]
char	source[32];	// Source of the attitude data[12],[23]
char	version[8];	// Version number (default = 1)
PGSt_double	startTime;	// Attitude dataset start time, secTAI93
PGSt_double	endTime;	// Attitude dataset end time, secTAI93
PGSt_real	interval;	// Expected Interval between records, SI second[13]
PGSt_uinteger	nURs;	// Number of input dataset universal references.
PGSt_uinteger	nRecords;	// Number of attitude records
PGSt_uinteger	eulerAngleOrder[3];	// Order of rotations as a permutation of 1=x, 2=y, 3=z
PGSt_real	qaParameters[16];	// Attitude data quality processing parameters
PGSt_real	qaStatistics[4];	// Quality assurance statistics
char	spare[280];	// Pad structure to 512 bytes

} PGSt_attitHeader;

[23] *This field is not reserved for the Toolkit and can be used for platform-specific data. For TRMM, it will contain an unsigned integer representing the TRMM ACS state. Any platform specific use should be documented.*

Table 7. Attitude Record Standard Structure

(One per record)

typedef struct
{

// Type	Name	Meaning
PGSt_double	secTAI93;	// Date and time as seconds from 1-1-93, secTAI93
PGSt_double	eulerAngle[3];	// Euler angle, radians
PGSt_double	angularVelocity[3];	// Angular rate about body, radians/s
PGSt_uinteger	qualityFlag;	// Attitude data quality flag
Char	spare[4];	// Pad structure to 64 bytes

} PGSt_attitRecord;

3. Summary of File Structures

The next two tables show how the headers, data records, and metadata fit into whole files.

Table 8. Overall Ephemeris File StructureRecord Type

Record Type	Record Declaration	Number of Records
Ephemeris Header	PGSt_ephemHeader	1
Universal References	char parentUR[256]	nURs (found in header record)
Ephemeris Records	PGSt_ephemRecord	nRecords (found in header record)
Orbit Metadata	PGSt_ephemMetadata	nOrbits (found in header record)

Table 9. Overall Attitude File Structure

Record Type	Record Declaration	Number of Records
Attitude Header	PGSt_attitHeader	1
Universal References	char parentUR[256]	nURs (found in header record)
Attitude Records	PGSt_attitRecord	nRecords (found in header record)

Appendix A: Use of Attitude Spares for the TRMM Spacecraft

Table 10. Attitude Header Implementation for TRMM (Metadata)

(One per file)

(Platform-specific use of spares is shown in *italics* near the end of the structure)

typedef struct

{

// Type	Name	Meaning
Char	spacecraftID[24];	// TRMM
Char	asciiTimeRange[48];	// Start and stop times to nearest hour, in ASCII
Char	source[32];	// Source of the attitude data
Char	version[8];	// Version number (default = 1)
PGSt_double	startTime;	// Attitude dataset start time, secTAI93
PGSt_double	endTime;	// Attitude dataset end time, secTAI93
PGSt_real	interval;	// Standard Interval between records, SI seconds
PGSt_uinteger	nURs;	// Number of input dataset universal references
PGSt_uinteger	nRecords;	// Number of attitude records
PGSt_uinteger	eulerAngleOrder[3];	// Order of rotations as a permutation of 1=x, 2=y, 3=z
PGSt_real	qaParameters[16];	// Attitude data quality processing parameters
PGSt_real	qaStatistics[4];	// Quality assurance statistics
PGSt_uinteger	*acsControlMode;*	*// Indicates use of gyros, Sun, maneuver, etc.*
PGSt_uinteger	*flyingModePriorInertial;*	*// Last flying mode before inertial lock (mode 0)*
PGSt_double	*quatOrb0ToECI[4];*	*// Quaternion from orbital to inertial at mode 0 lock*
char	*spares[240];*	*// Pad structure to 512 bytes*

} DpTPrAttitudeHeader;

It is to be emphasized that the SDP Toolkit does not access the spares used here; they are used by the preprocessing system for chaining coordinate system transformations peculiar to the TRMM spacecraft. It is, of course, desirable to document the use of spares as shown above in italics, and in accompanying documentation.

Appendix B: Use of Attitude Spares for the Terra, Aqua, and Aura Spacecrafts

Table 11. Attitude Header Implementation for Terra, Aqua, and Aura (Metadata)

(One per file)

(Platform-specific use of spares is shown in *italics* near the end of the structure)

typedef struct

{

// Type	Name	Meaning
Char	spacecraftID[24];	// EOSPM1 or EOSAURA
Char	asciiTimeRange[48];	// Start and stop times to nearest hour, in ASCII
Char	source[32];	// Source of the attitude data
Char	version[8];	// Version number (default = 1)
PGSt_double	startTime;	// Attitude dataset start time, secTAI93
PGSt_double	endTime;	// Attitude dataset end time, secTAI93
PGSt_real	interval;	// Standard Interval between records, SI seconds
PGSt_uinteger	nURs;	// Number of input dataset universal references
PGSt_uinteger	nRecords;	// Number of attitude records
PGSt_uinteger	eulerAngleOrder[3];	// Order of rotations as a permutation of 1=x, 2=y, 3=z
PGSt_real	qaParameters[16];	// Attitude data quality processing parameters
PGSt_real	qaStatistics[4];	// Quality assurance statistics
PGSt_real	*psQaStatistics[4];*	*// Additional quality assurance statistics*
PGSt_real	*psQaParameters[16];*	*// Additional attitude data quality processing parameters*
Char	*spares[200];*	*// Pad structure to 512 bytes*

} DpTPrAttitudeHeader;

It is to be emphasized that the SDP Toolkit does not access the spares used here; they are used by the preprocessing system for chaining coordinate system transformations peculiar to the TRMM spacecraft. It is, of course, desirable to document the use of spares as shown above in italics, and in accompanying documentation.

Appendix C: Use of Ephemeris Spares for the Terra, Aqua, and Aura Spacecraft

Table 12. Ephemeris Header Implementation for Terra, Aqua, and Aura (Metadata) (1 of 2)

(One per file)

(Platform-specific use of spares is shown in *italics* near the end of the structure)

typedef struct

{

// Type	Name	Meaning
Char	spacecraftID[24];	// Spacecraft Name
Char	asciiTimeRange[48];	// Start stop times to nearest hour or better, in ASCII
Char	source[32];	// Source of the data
Char	version[8];	// Version number (default = 1)
PGSt_double	startTime;	// Ephemeris dataset start time, secTAI93
PGSt_double	endTime;	// Ephemeris dataset end time, secTAI93
PGSt_real	interval;	// Expected interval between records, SI seconds
PGSt_uinteger	nURs;	// Number of input dataset universal references
PGSt_uinteger	nRecords;	// Number of ephemeris records
PGSt_uinteger	nOrbits;	// Number of orbits spanned, including fragments
PGSt_uinteger	orbitNumberStart;	//Number of first orbit or part orbit in file
PGSt_uinteger	orbitNumberEnd;	//Number of last orbit or part orbit in file
Char	keplerRefFrame[8];	// Reference Frame: e.g. "TOD", "TOR" or "J2000" of the Keplerian Elements

Table 12. Ephemeris Header Implementation for Terra, Aqua, and Aura (Metadata) (2 of 2)

// Type	Name	Meaning
PGSt_double	keplerElements[6];	// Osculating Keplerian elements at epoch
PGSt_double	keplerEpochTAI;	// TAI 93 Epoch of the Reference Frame
PGSt_real	qaParameters[16];	// Ephemeris data quality processing parameters
PGSt_real	qaStatistics[4];	// Quality assurance statistics
PGSt_double	*orbitalPeriod;*	*// Terra orbital period*
PGSt_double	*descNodePropagation;*	*// Change in descending node crossing longitude between successive crossings*
PGSt_uinteger	*fddReplacement;*	*// Status of FDD replacement*[24]
Char	*spares[196];*	*// Pad to 512 bytes*

} DpTPrEphemerisHeader;

It is to be emphasized that the SDP Toolkit does not access the spares used here; they are used by the preprocessing system for chaining orbit metadata and FDD dataset replacement status peculiar to the Terra, Aqua, and Aura spacecraft.

[24] 0 = no FDD replacement required, 1 = FDD replacement requested, 2 = FDD replacement achieved.

Appendix M. Problem Identification List

The list of known problems as of December 2017 for the SCF Toolkit 5.2.20 delivery of the SDP Toolkit can be found in section 5 of the SDP Toolkit 5.2.20 Version Description Document (VDD) for the ECS Project.

This page intentionally left blank.

Appendix N. Structure of the File "utcpole.dat”

The file specification given here is not expected to change for the life of the EOSDIS project. It is provided so that users may read columns other than those read by the Toolkit. The Toolkit reads only the first header line of this file and columns 1,2,4,6,7,and 8. The columns are as follows:

1. modified UTC Julian date

2. x component of polar motion, arc seconds

3. one standard deviation error estimate for column 2 values (see qualification below)

4. y component of polar motion

5. one standard deviation error estimate for column 4 values (see qualification below)

6. UT1 - UTC in seconds of time

7. one standard deviation error estimate for column 6 values (see qualification below)

8. data quality indicator

The columns are tab delimited. There are exactly 65 characters per line, including the newline character, except in the header. The two header lines total 168 characters, including the newlines. The data are all from the U.S. Naval Observatory (USNO), except for the error values from 1972 (beginning of file) to 1979; these are guesses by Dr. Peter Noerdlinger in the absence of other information, but were sent to the Observatory for comment and no objection was received. The errors after 1979 Jan 1 are one standard deviation errors and could easily be read by users who need these numbers. There was no project requirement for accuracy, but the Toolkit staff felt that the numbers should be saved in case of later interest. Date flagged "f" in the last column are "final" but may change by very small amounts (cm to mm range), when new data are ingested at USNO or the Observatory updates their earth rotation model. The data marked "p" are predicted data. They tend to change more as updates are performed by the USNO.

Selected sections of a typical data file are shown below. The regions given in detail are beginning of file, a section around a leap second, the transition to predicted data, and the end of the file.

```
File Updated: 1998-03-05T17:26:41Z, using USNO ser7 finals.data file of Mar  5

MJD       x(arc sec)    x error      y(arc sec)   y error      UT1-UTC(s)  UT error     qual

41317        +0.061000    0.002000     +0.051000    0.002000     -0.043200    0.000200     f

41318        +0.058000    0.002000     +0.049000    0.002000     -0.046100    0.000200     f

41319        +0.055000    0.002000     +0.048000    0.002000     -0.049000    0.000200     f

41320        +0.052000    0.002000     +0.047000    0.002000     -0.052000    0.000200     f

41321        +0.048000    0.002000     +0.045000    0.002000     -0.054900    0.000200     f

41322        +0.045000    0.002000     +0.044000    0.002000     -0.057900    0.000200     f

----------------section removed here covering many decades, to save space-----------------

-----------------next few lines show transition at a leap second-------------------------

50077        -0.164345    0.000052     +0.174418    0.000129     -0.429816    0.000010     f

50078        -0.166356    0.000052     +0.177657    0.000130     -0.432590    0.000002     f

50079        -0.168543    0.000059     +0.180703    0.000099     -0.435312    0.000011     f

50080        -0.170630    0.000055     +0.183521    0.000088     -0.437914    0.000011     f

50081        -0.172500    0.000054     +0.186204    0.000088     -0.440347    0.000011     f

50082        -0.174396    0.000107     +0.188956    0.000130     -0.442584    0.000038     f

50083        -0.176051    0.000119     +0.191918    0.000124     +0.555381    0.000022     f

50084        -0.177290    0.000118     +0.194805    0.000120     +0.553526    0.000020     f

50085        -0.178255    0.000098     +0.197606    0.000157     +0.551818    0.000015     f

--------------section removed here covering over twoyears, to save space----------------

-----------------next few lines show transition to predicted data-----------------------

50868        -0.051310    0.000209     +0.187877    0.000224     +0.115291    0.000015     f

50869        -0.054006    0.000216     +0.188612    0.000245     +0.113184    0.000016     f

50870        -0.056066    0.000180     +0.189348    0.000237     +0.110919    0.000016     f

50871        -0.057614    0.000176     +0.190131    0.000231     +0.108499    0.000017     f

50872        -0.058668    0.000158     +0.191538    0.000239     +0.105943    0.000017     f

50873        -0.059457    0.000106     +0.193336    0.000270     +0.103315    0.000027     f

50874        -0.060498    0.000096     +0.195182    0.000176     +0.100719    0.000031     f

50875        -0.061903    0.000069     +0.196987    0.000150     +0.098242    0.000031     f
```

50876	-0.063387	0.000076	+0.198881	0.000169	+0.095935	0.000038	f
50877	-0.064763	0.004200	+0.200551	0.004200	+0.093803	0.000300	p
50878	-0.066208	0.005100	+0.202151	0.005100	+0.091816	0.000505	p
50879	-0.067709	0.005713	+0.203691	0.005713	+0.089933	0.000684	p
50880	-0.069255	0.006192	+0.205182	0.006192	+0.088073	0.000849	p
50881	-0.070836	0.006591	+0.206632	0.006591	+0.086174	0.001004	p
50882	-0.072444	0.006936	+0.208049	0.006936	+0.084217	0.001152	p
50883	-0.074071	0.007242	+0.209440	0.007242	+0.082200	0.001293	p
50884	-0.075711	0.007518	+0.210811	0.007518	+0.080116	0.001429	p
50885	-0.077358	0.007770	+0.212168	0.007770	+0.077970	0.001561	p
50886	-0.079007	0.008003	+0.213516	0.008003	+0.075777	0.001690	p
50888	-0.082293	0.008422	+0.216201	0.008422	+0.071295	0.001938	p
50889	-0.083923	0.008613	+0.217545	0.008613	+0.069030	0.002058	p
50890	-0.085540	0.008794	+0.218895	0.008794	+0.066774	0.002175	p
50891	-0.087141	0.008965	+0.220254	0.008965	+0.064551	0.002291	p
----------	numerous lines removed here, to save space						----------
50959	-0.126231	0.014474	+0.345196	0.014474	-0.074207	0.008276	p
50960	-0.125711	0.014523	+0.347152	0.014523	-0.075710	0.008351	p
50961	-0.125162	0.014571	+0.349097	0.014571	-0.077087	0.008425	p

This page intentionally left blank.

Abbreviations and Acronyms

A.A.	Astronomical Almanac
AA	ancillary data access
ACS	Attitude Control System
AI&T	algorithm integration & test
AIRS	Atmospheric Infrared Sounder
AM	see EOSAM
API	application program interface
APID	application process identifier
Aqua	EOS PM Project Spacecraft 1, afternoon spacecraft series – AIRS, AMSR-E, AMSU, CERES, HSB, MODIS instruments; formerly PM-1
Aura	EOS Project afternoon spacecraft series; HIRDLS, MLS, OMI, and TES instruments; formerly CHEM
ASCII	American Standard Code for Information Interchange
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
B1950	Mean Celestial Reference Frame at JD 2433282.0 TDT (2433282.0 is noon, not midnight, and is equivalent to 1949-12-31T22:09:46.816 UTC)
BNF	Backus–Naur Form
CBP	celestial body position
CCR	configuration change request
CCSDS	Consultative Committee on Space Data Systems
CDRL	Contract Data Requirements List
CDS	CCSDS day segmented time code
CERES	Clouds and Earth Radiant Energy System
CM	configuration management
COTS	commercial off–the–shelf software
CRC	cyclic redundancy code
CSC	coordinate system conversion
CSMS	Communications and Systems Management Segment (ECS)

CUC	constant and unit conversions
CUC	CCSDS unsegmented time code
DAAC	distributed active archive center
DBMS	database management system
DCE	distributed computing environment
DCW	Digital Chart of the World
DDF	data distribution facility (Pacor)
DEM	digital elevation model
DPFT	Data Processing Focus Team
DPREP	Data Preprocessing
DTM	digital terrain mode
ECI	Earth centered inertial
ECR	Earth centered rotating
ECS	EOSDIS Core System
EDC	Earth Resources Observation Systems (EROS) Data Center
EDHS	ECS Data Handling System
EDOS	EOSDIS Data and Operations System
EMOS	EOS Mission Operations System
EOS	Earth Observing System
EOSAM	EOS AM Project (morning equator crossing spacecraft series)
EOSDIS	Earth Observing System Data and Information System
EOSPM	EOS PM Project (afternoon equator crossing spacecraft series)
EPH	ephemeris data access
ESDIS	Earth Science Data and Information System (GSFC Code 505)
ET	ephemeris tool
FDD	Flight Dynamics Division
FDF	flight dynamics facility
FNOC	Federal Naval Operations Center
FOV	field of view
ftp	file transfer protocol

GAST	Greenwich apparent sidereal time
GCT	geo–coordinate transformation
GCTP	general cartographic transformation package
GIS	geographic information systems
GMST	Greenwich mean sidereal time
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
GTDS	Goddard Trajectory Determination System
HDF	hierarchical data format
HITC	Hughes Information Technology Corporation
HOM	Hotine Oblique Mercator
http	hypertext transport protocol
I&T	integration & test
I/O	input/output
IAU	International Astronomical Union
ICD	interface control document
IDL	interactive data language
IEEE	Institute of Electrical and Electronic Engineers
IERS	International Earth Rotation Service
IMS	information management system
IP	Internet protocol
IWG	Investigator Working Group
J2000	Mean Celestial Reference Frame at JD 2451545.0 TDT (2451545.0 is noon, not midnight, and is equivalent to 2000-01-01T11:58:55.816 UTC)
JNC	jet navigational charts
JPL	Jet Propulsion Laboratory
L0	Level 0 (zero)
LaRC	Langley Research Center
LIS	Lightening Imaging Sensor
M&O	maintenance and operations
MCF	metadata configuration file

MDU	missing data unit
MDUE	Missing Data Unit Entry
MDUL	missing data unit list
MEM	memory management
MET	metadata
MODIS	Moderate–Resolution Imaging Spectroradiometer
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputer Applications
netCDF	network common data format
NGDC	National Geophysical Data Center
NMC	National Meteorological Center (NOAA)
ODL	object description language
PACOR	packet processor
PC	process control
PCF	process control file
PDPS	planning & data production system
PDR	Preliminary Design Review
PDS	production data set
PGE	product generation executive (formerly product generation executable)
PGS	Product Generation System
PGSTK	Product Generation System Toolkit
PM	see EOSPM
POSIX	Portable Operating System Interface for Computer Environments
QA	quality assurance
QAC	quality and accounting capsule
RDBMS	relational database management system
RPC	remote procedure call
RRDB	recommended requirements database
SCF	Science Computing Facility

SDP	science data production
SDPF	science data processing facility
SDPS	Science Data Processing Segment (ECS)
SES	scheduling and execution subsystem
SFDU	standard formatted data unit
SGI	Silicon Graphics Incorporated
SI	systeme international
SM & A	Steven Myers and Associates
SMF	status message file
SMAP	Soil Moisture Active Passive
SMP	Symmetric Multi–Processing
SOM	Space Oblique Mercator
SPCS	State Plane Coordinates Spheroid
SPSO	Science Processing Support Office
SSM/I	Special Sensor for Microwave/Imaging
TAI	Temps Atomique International (International Atomic Time)
TBD	to be determined
TD	time date conversion
TDB	Barycentric Dynamical Time
TDRSS	Tracking and Data Relay Satellite System
TDT	Terrestrial Dynamical Time
Terra	EOS AM Project Spacecraft 1, morning spacecraft series – ASTER, CERES, MISR, MODIS and MOPITT instruments; formerly AM-1
THG	The HDF Group
TLCF	team leader computing facility
TOD	True of Date
TONS	TDRSS On Board Navigational System
TOR	True of Reference
TRMM	Tropical Rainfall Measuring Mission (joint US – Japan)
TSS	(TDRSS) Service Session
UARS	Upper Atmosphere Research Satellite

UCAR	University Corporation for Atmospheric Research
UR	Universal Reference
URL	universal reference locator
USDC	United States Department of Commerce
USNO	United States Naval Observatory
UT	universal time
UTC	Coordinated Universal Time
UTCF	universal time correlation factor
UTM	universal transverse mercator
VCDU	virtual channel data unit
VPF	vector product format
WWW	World Wide Web