

170-TP-012-003

A Data Formatting Toolkit for Extended Data Providers to NASA's Earth Observing System Data and Information System (V4.0)

March 2001

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Abe Taaheri, and Phuong Nguyen
EOSDIS Core System Project

Date

SUBMITTED BY

Darryl Arrington, Manager Toolkit
Larry Klein, Manager Toolkit
EOSDIS Core System Project

Date

Raytheon Company
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document describes the Metadata and Time/Date (MTD) Tools that are extracted from the Science Data Processing (SDP) Toolkit for the Earth Observing System Data and Information System (EOSDIS) Core System (ECS) Project to comprise the MTD Toolkit release. The extracted tools are to be called “Toolkit”, “Toolkit_MTD”, or "the Tools" in the rest of this document (the SDP Toolkit that the present tools have been extracted from will still be referenced to as “SDP Toolkit”).

The tools in the Toolkit_MTD have the same functionality as the respective tools in the SDP Toolkit; with some minor differences in error reporting; and some added functionality such as enabling reading metadata from an ASCII file. The differences are described in this document. The SDP Toolkit is described in the *Release 6A SDP Toolkit User's Guide for the ECS Project (333-CD-600-001)*.

Toolkit_MTD is intended to be used in conjunction with the Hierarchical Data Format (HDF) tools and HDF-EOS extensions to HDF. These tools are described in *HDF-EOS Library User's Guide for the ECS Project, Volume 1: Overview and Examples (170-TP-600-001)* and in *HDF-EOS Library User's Guide for the ECS Project, Volume 2: Function Reference Guide (170-TP-601-001)*. Both HDF4 and HDF5 - based files are supported.

The primary purpose of this tool set is to allow EOSDIS extended data providers the capability of formatting their products in ECS standard formats, without requiring the entire SDP Toolkit package. Toolkit_MTD will also allow creation of and access to ECS standard metadata. Toolkit_MTD is intended to be used by data providers who will produce products at their local facilities and then deliver those products to be archived and distributed from ECS Data Active Archive Centers (DAACs). The tools are not intended to be used by ECS instrument teams who will deliver production code to the DAACs. For this reason, the metadata and time conversion tools have been extracted from the rest of the Toolkit, giving a streamlined version for use by external (to ECS) data providers.

Toolkit_MTD will operate in Windows 98 & NT platforms as well as on the Unix and LINUX platforms which the SDP Toolkit currently operates on (see Note in Section 5.1 for an update on this). The software is written in the C language and FORTRAN bindings are provided. Toolkit_MTD will be kept current with updates to the SDP toolkit.

This document provides a listing of routine calling sequences, detailed descriptions, examples of usage as lieu as installation instructions. A brief description of software design and content is also provided. The document accompanies a software delivery that contains implementations of the tools described therein. We note that this version of extracted tools contains provisions for error/status messaging, process control, and file handling in lieu of an operational data production system. This handling will be via manipulation of and access to external files.

Technical Points of Contact are:

Larry Klein, larry@eos.hitc.com

Abe Taaheri, ataaheri@eos.hitc.com

Emergent Information Technologies, Inc.

9315 Largo Dr. West, Suite 250

Largo, MD 20774

Abstract

The MTD Toolkit Users Guide for the ECS Project is an extraction of and an extension of SDP Toolkit Users Guide for the ECS delivered under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract (NAS5-60000). It was first delivered in October 1998. This current Users Guide is updated to match the Release 6A SDP Toolkit delivery. The MTD Toolkit Users Guide describes MTD Toolkit routine usage for EOSDIS extended data providers, who will produce products at their local facilities and then deliver those products to be archived and distributed from the ECS Data Active Archive Centers (DAACs). The MTD Toolkit allows EOSDIS extended data providers the capability of formatting their products in ECS standard formats, without requiring the entire SDP Toolkit package. This document describes the overall design of the MTD 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.

Keywords: toolkit, metadata, MTD, HDF, HDF5, HDF-EOS, data, format, production, error, handling, input, output, windows 98, NT, LINUX

This page intentionally left blank.

Contents

Preface

1. Introduction

1.1	Identification	1-1
1.2	Scope.....	1-1
1.3	Purpose and Objectives.....	1-1
1.4	Status and Schedule	1-2
1.5	Document Organization	1-4

2. Related Documentation

2.1	Parent Documents	2-1
2.2	Applicable Documents.....	2-1
2.3	Information Documents	2-1

3. Toolkit Design

4. Toolkit Usage and Functionality

4.1	Introduction.....	4-1
4.2	Functionality	4-1
4.3	MET Tools	4-1
4.4	Time/Date Tools	4-2
4.5	Error/Status Log.....	4-2
4.6	HDF-EOS	4-2

5. Toolkit_MTD Installation and Maintenance

5.1	Introduction.....	5-1
5.2	Installation Procedures for UNIX Platforms	5-1
5.2.1	Toolkit_MTD Release Notes	5-1
5.2.2	To Install the Toolkit_MTD from a Disk-Based Tar File.....	5-3
5.2.3	Compiling User Code with the Toolkit_MTD.....	5-23
5.3	Installation Procedures for Windows NT/98	5-25
5.4	Instructions on Making Changes to Installation Procedures for UNIX Platforms	5-30
5.5	Link Instructions	5-32
5.6	Test Drivers.....	5-33
5.7	User Feedback Mechanism.....	5-34

6. Toolkit Specification

6.1	Introduction.....	6-1
6.2	Toolkit Tools.....	6-2
6.2.1	Metadata Tools	6-2
6.2.2	Error/Status Reporting (SMF Tools)	6-41
6.2.3	Time and Date Conversion Tools	6-52

List of Figures

3-1.	Diagram showing the connection between MET tools and support tools	3-2
3-2.	Diagram showing the relation between TD tools and support tools	3-3
3-3.	Data Flow Diagram	3-4

List of Tables

1-1.	Toolkit Routine Key	1-2
1-2.	Toolkit Routine Listing	1-3
5-1.	SDP Toolkit_MTD Development Configuration	5-21

5-2. Required Directory Environment Variables	5-23
5-3. Required Compiler and Library Environment Variables	5-23
5-4. Values of OSTYPE.....	5-31
5-5. Environment Variables	5-31
6-1. PGS_MET_SetFileId Returns	6-4
6-2. PGS_MET_GetFileId Inputs	6-6
6-3. PGS_MET_GetFileId Returns.....	6-6
6-4. PGS_MET_SDstart Inputs	6-8
6-5. PGS_MET_SDstart Ouputs.....	6-8
6-6. PGS_MET_SDstart Returns	6-9
6-7. PGS_MET_SDend Outputs.....	6-10
6-8. PGS_MET_SDend Returns	6-10
6-9. PGS_MET_Init Inputs	6-12
6-10. PGS_MET_Init Outputs	6-12
6-11. PGS_MET_Init Returns	6-13
6-12. PGS_MET_Init_NonMCF Inputs	6-16
6-13. PGS_MET_Init_NonMCF Outputs.....	6-16
6-14. PGS_MET_Init_NonMCF Returns	6-17
6-15. PGS_MET_SetAttr Inputs.....	6-20
6-16. PGS_MET_SetAttr Returns	6-21
6-17. PGS_MET_GetSetAttr Inputs	6-25
6-18. PGS_MET_GetSetAttr Outputs	6-25
6-19. PGS_MET_GetSetAttr Returns.....	6-26
6-20. PGS_MET_GetPCAttr Inputs	6-28
6-21. PGS_MET_GetPCAttr Outputs.....	6-29
6-22. PGS_MET_GetPCAttr Returns.....	6-29
6-23. PGS_MET_GetConfigData Inputs	6-33
6-24. PGS_MET_GetConfigData Outputs	6-33

6-25. PGS_MET_GetConfigData Returns.....	6-34
6-26. PGS_MET_Write Inputs	6-36
6-27. PGS_MET_WriteReturns.....	6-37
6-28. PGS_SMF_SetStaticMsg Returns	6-42
6-29. PGS_SMF_SetDynamicMsg Returns.....	6-44
6-30. PGS_SMF_GetMsgByCode Returns	6-47
6-31. PGS_SMF_TestStatusLevel Returns.....	6-49
6-32. Estimated Errors in UT1 Predictions (Milliseconds of Time and Equivalent Meters of Geolocation Error)	6-58
6-33. PGS_TD_SetFileId Returns	6-60
6-34. PGS_TD_UTCtoTAI Inputs.....	6-62
6-35. PGS_TD_UTCtoTAI Outputs	6-62
6-36. PGS_TD_UTCtoTAI Returns	6-63
6-37. PGS_TD_TAItoUTC Inputs.....	6-65
6-38. PGS_TD_TAItoUTC Outputs	6-65
6-39. PGS_TD_TAItoUTC Returns	6-65
6-40. PGS_TD_TAItoTAIjd.c Inputs	6-67
6-41. PGS_TD_TAItoTAIjd Outputs	6-67
6-42. PGS_TD_TAIjdtoTAI Inputs.....	6-69
6-43. PGS_TD_TAItoGAST Inputs	6-71
6-44. PGS_TD_TAItoGAST Outputs.....	6-71
6-45. PGS_TD_TAItoGAST Returns.....	6-71
6-46. PGS_TD_UTCtoSctime Returns.....	6-74
6-47. PGS_TD_Sctime_to_UTC Outputs.....	6-77
6-48. PGS_TD_Sctime_to_UTC Returns.....	6-77
6-49. PGS_TD_ASCIItime_AtoB Inputs	6-80
6-50. PGS_TD_ASCIItime_AtoB Outputs.....	6-80
6-51. PGS_TD_ASCIItime_AtoB Returns.....	6-80
6-52. PGS_TD_ASCIItime_BtoA Inputs	6-82

6-53. PGS_TD_ASCIItime_BtoA Outputs.....	6-82
6-54. PGS_TD_ASCIItime_BtoA Returns.....	6-82
6-55. PGS_TD_UTCtoGPS Inputs	6-84
6-56. PGS_TD_UTCtoGPS Outputs	6-84
6-57. PGS_TD_UTCtoGPS Returns.....	6-84
6-58. PGS_TD_GPSstoUTC Inputs	6-86
6-59. PGS_TD_GPSstoUTC Outputs	6-86
6-60. PGS_TD_GPSstoUTC Returns.....	6-86
6-61. PGS_TD_UTCtoTDTjed Inputs.....	6-88
6-62. PGS_TD_UTCtoTDTjed Outputs	6-88
6-63. PGS_TD_UTCtoTDTjed Returns	6-88
6-64. PGS_TD_UTCtoTDBjed Inputs.....	6-91
6-65. PGS_TD_UTCtoTDBjed Outputs.....	6-91
6-66. PGS_TD_UTCtoTDBjed Returns	6-91
6-67. PGS_TD_TimeInterval Inputs.....	6-94
6-68. PGS_TD_TimeInterval Outputs.....	6-94
6-69. PGS_TD_TimeInterval Returns	6-94
6-70. PGS_TD_UTCtoUTCjd Inputs	6-96
6-71. PGS_TD_UTCtoUTCjd Outputs.....	6-96
6-72. PGS_TD_UTCtoUTCjd Returns.....	6-96
6-73. PGS_TD_UTCjdtoUTC Inputs	6-98
6-74. PGS_TD_UTCjdtoUTC Outputs.....	6-98
6-75. PGS_TD_UTCjdtoUTC Returns.....	6-98
6-76. PGS_TD_UTCtoUT1 Inputs	6-100
6-77. PGS_TD_UTCtoUT1 Outputs	6-100
6-78. PGS_TD_UTCtoUT1jd Inputs.....	6-103
6-79. PGS_TD_UTCtoUT1jd Outputs	6-103
6-80. PGS_TD_UTCtoUT1jd Returns.....	6-103

6-81. Get Leap Second Inputs.....	6-105
6-82. Get Leap Second Outputs.....	6-105
6-83. Get Leap Seconds Returns.....	6-106

Appendix A. Assumptions

Appendix B. SMF Usage

Appendix C. PCFT File

Appendix D. Population of Granule Level Metadata Using the Metadata Tools

Appendix E. Test Drivers

Appendix F. Config File Used by MET/TD Tools

Appendix G. Structure of the File “utcpole.dat”

Abbreviations and Acronyms

1. Introduction

1.1 Identification

This Toolkit Users Guide is an extraction of and an extension to the SDP Toolkit Users Guide for the ECS delivered under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract (NAS5-60000). The current SDP Toolkit Users Guide is updated for the Release 6A Toolkit delivery made in November 2000. Subsequent versions will accompany major ECS releases. This Toolkit Users Guide will be updated in conjunction with the SDP Toolkit Users Guide

1.2 Scope

This User's Guide describes software tools which can be used by data providers who will produce products at their local institutions and then deliver those products to ECS DAACs for archival and distribution. The user calling interface is the same as that contained in the SDP Toolkit version 5.2.7 (*Release 6A SDP Toolkit User's Guide for the ECS Project, 333-CD-600-001*). The tools described in this document consist of metadata formatting and access tools and time and date conversion tools.

It is expected that users of this software will use it in conjunction with HDF and HDF-EOS data formatting and access software. HDF is the NASA ECS Project standard data format and HDF-EOS is an extension of that format, focusing HDF data structure standards on specific earth sciences data types. Users will use the metadata tools to build ECS standard metadata which will be included as global attributes with in the HDF (HDF-EOS) data granules.

The SDP Toolkit and HDF-EOS use as an underlying time format, TAI, or International Atomic Time. For this reason, time and data conversion tools are provided in this package so that users can create other time formats of their choosing.

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 describes in detail the installation and usage of metadata access and time conversion tools. A user will be provided with detailed calling sequences and examples of usage of all the routines described in this document. Descriptions of error handling and external file access are also provided. Instructions for access to the software and electronic versions of this document will be provided.

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 February 2001. Table 1–2 below gives a complete listing; brief description; and delivery dates of Toolkit software available to users.

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
MET	Meta Data Access	6.2.1
SMF	Status Message File (Error/Status)	6.2.2
TD	Time Date Conversion	6.2.3

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 (e.g., PGS_MET).

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

Tool Name	Description	Date	Page
PGS_MET_GetConfigData	Enables the user to get the values of Config data parameters held in the PC table		6-33
PGS_MET_GetFileId	This tool retrieves logical ID assigned for a file entry in PCFT file filetable.temp. It returns FileId if successful, 0 otherwise.		6-6
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		6-28
PGS_MET_GetSetAttr	The MCF is initialized into memory, some parameters are automatically set and some are set using PGS_MET_SetAttr. This tool retrieves these values		6-25
PGS_MET_Init	Initializes a metadata configuration file (MCF)		6-12
PGS_MET_Init_NonMCF	Initializes an ASCII file containing metadata.		6-16
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		6-40
PGS_MET_SetAttr	Enables the user to set the value of metadata parameters		6-20
PGS_MET_SetFileId	This tool sets logical IDs assigned for the user defined files in PCFT file filetable.temp (See Note below)		6-4
PGS_MET_SDstart	This tool opens HDF4 or HDF5 files for writing metadata to them		6-8
PGS_MET_SDeud	This tool closes files opened by PGS_MET_SDstart		6-10
PGS_MET_Write	Enables the user to write different groups of metadata to separate HDF attributes		6-36
PGS_SMF_GetMsg	Provide the means to retrieve a previously set message from the static buffer PGS_SMF_Set...		6-48
PGS_SMF_GetMsgByCode	Provide the means to retrieve the message string corresponding to a specific mnemonic code		6-47
PGS_SMF_SetDynamicMsg	Provide the means to set a user-defined error/status message in response to the outcome of some segment of processing.		6-44
PGS_SMF_SetStaticMsg	Provide the means to set a predefined error/status message in response to the outcome of some segment of processing.		6-42
PGS_SMF_TestStatusLevel	Will return a defined status level constant		6-49
PGS_TD_SetFileId	This tool sets logical Ids assigned for the files for TD tool in the PCFT file filetable.temp (See Note below)		6-60
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		6-80
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		6-82
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		6-86
PGS_TD_LeapSec	Find Leap second value		6-105
PGS_TD_Sctime_to_UTC	Converts spacecraft clock time to UTC for EOS platforms or for foreign spacecraft		6-76
PGS_TD_TAtoGAST	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)		6-71

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

Tool Name	Description	Date	Page
PGS_TD_TAIjdtotAI	Converts TAI Julian date to time in TAI seconds since 12 AM UTC 1-1-1993.		6-69
PGS_TD_TAItoTAIjd	Converts time in TAI seconds since 12 AM UTC 1-1-1993 toTAI Julian date.		6-67
PGS_TD_TAItoUTC	Converts TAI time value to UTC time		6-65
PGS_TD_TimeInterval	Computes the elapsed TAI time in seconds between any two epochs after January 1, 1958		6-94
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		6-84
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		6-62
PGS_TD_UTCtoTDBjed	UTC to Barycentric Dynamical Time (TDB) time conversion		6-91
PGS_TD_UTCtoTDTjed	UTC to Terrestrial Dynamical Time (TDT) time conversion		6-88
PGS_TD_UTCtoUT1	Converts UTC to UT1 time		6-100
PGS_TD_UTCtoUT1jd	Converts UTC time in CCSDS ASCII Time Code to UT1 time as a Julian date		6-103
PGS_TD_UTCjdtotUTC	Converts UTC as a Julian date to UTC in CCSDS ASCII Time Code A format.		6-98
PGS_TD_UTCtoUTCjd	Converts UTC in CCSDS ASCII Time Code A format to UTC as a Julian date.		6-96
PGS_TD_UTC_to_Sctime	Converts UTC to Spacecraft clock time for EOS standard of Foreign Spacecraft		6-73

Note: If both the MET and TD tools are to be used in the same code, all input/output entries must be in one **filetable.temp**. One call to PGS_MET_SetFileId or PGS_TD_SetFileId will be enough to load Ids and physical file names into memory.

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

Section 6	Toolkit Specification—Contains calling sequences, description and usage examples for Toolkit routines.
Appendix A	Assumptions
Appendix B	SMF Usage
Appendix C	PCFT File
Appendix D	Population of Granule Level Metadata using the SDP metadata tools
Appendix E	Test Drivers
Appendix F	Config File Used by MET/TD Tools
Appendix G	Structure of the File "utcpole.dat"
Acronyms and Abbreviations	

This page intentionally left blank.

2. Related Documentation

2.1 Parent Documents

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

333-CD-600 Release 6A SDP Toolkit User's Guide for the ECS Project.
Available at <http://edhs1.gsfc.nasa.gov> under Toolkits

2.2 Applicable Documents

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

170-TP-600 HDF-EOS Library User's Guide for the ECS Project, Volume 1:
Overview and Examples . Available at <http://edhs1.gsfc.nasa.gov>
under Toolkits.

170-TP-601 HDF-EOS Library User's Guide for the ECS Project, Volume 2:
Function Reference Guide. . Available at <http://edhs1.gsfc.nasa.gov>
under Toolkits.

CCSDS 301.0-B-2 Consultative Committee for Space Data Systems (CCSDS)
Recommendation for Space Data System Standards: Time Code
Formats, 4/90

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
This is also available via anonymous file transfer protocol (ftp) from
<ftp.ncsa.uiuc.edu> (141.142.20.50)

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.

194-815-SI4 SDP Toolkit Primer (current version available through WWW access:
<http://edhs1.gsfc.nasa.gov>)

This page intentionally left blank.

3. Toolkit Design

The Metadata and Time/Date tools have been extracted from the SDP Toolkit. The Toolkit_MTD's design follows the design of the SDP Toolkit with two major differences; the first replaces the process control file with a simple **filetable** file and the second is using a different scheme, but with similar format, in reporting toolkit errors. The PCF is an external file which maps logical unit numbers to physical handles. In ECS, this file is generated automatically prior to data production. Filetable is used for the same purpose but its usage is left up to the user- to be manually or automatically generated. This is explained in detail in sections 4, 5, 6, and Appendices.

As in the SDP Toolkit, 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 tools. 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").

The Toolkit routines are divided into two classes; Main tools and Support tools.

The Main tools are tools that the users interact with such as the TD tools, MET tools, and several optional SMF tools for handling error messages.

The Support tools are the routines that are used by the main tools and will not be called by the user. These are the IO, PC, CUC, CSC, CBP, MEM, and several MET routines. The diagrams on the next few pages show the inter relation between main and support tools and the data flow.

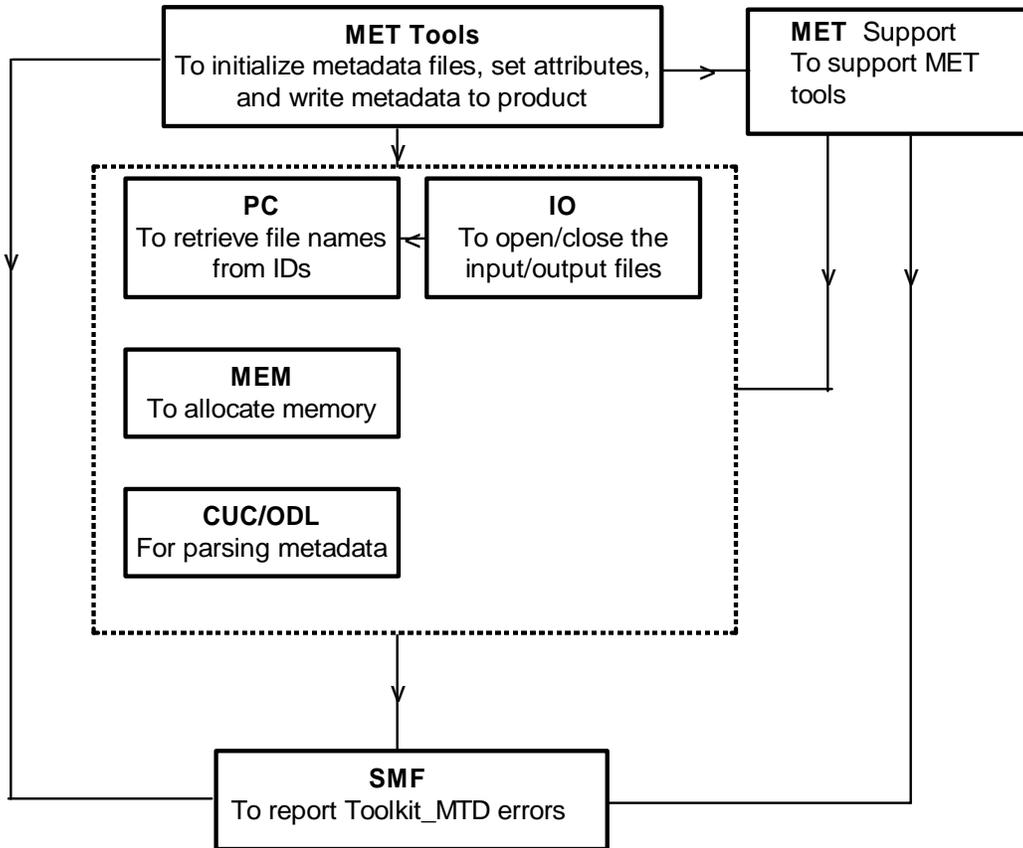


Figure 3-1. Diagram showing the connection between MET tools and support tools

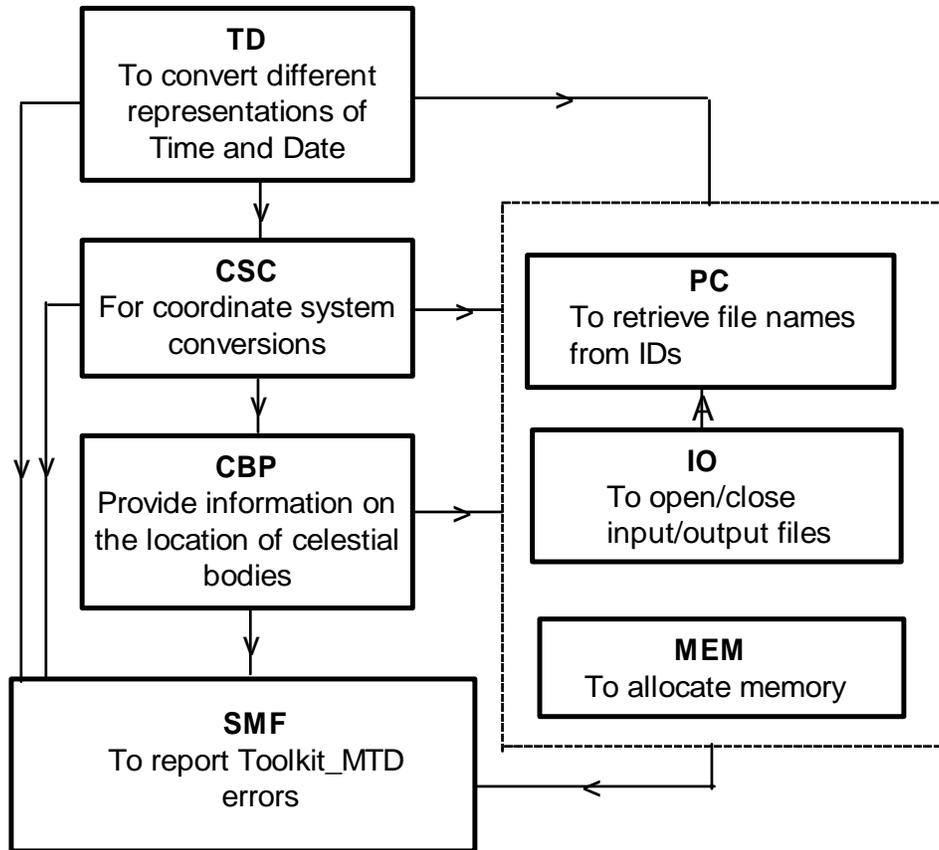


Figure 3-2. Diagram showing the relation between TD tools and support tools

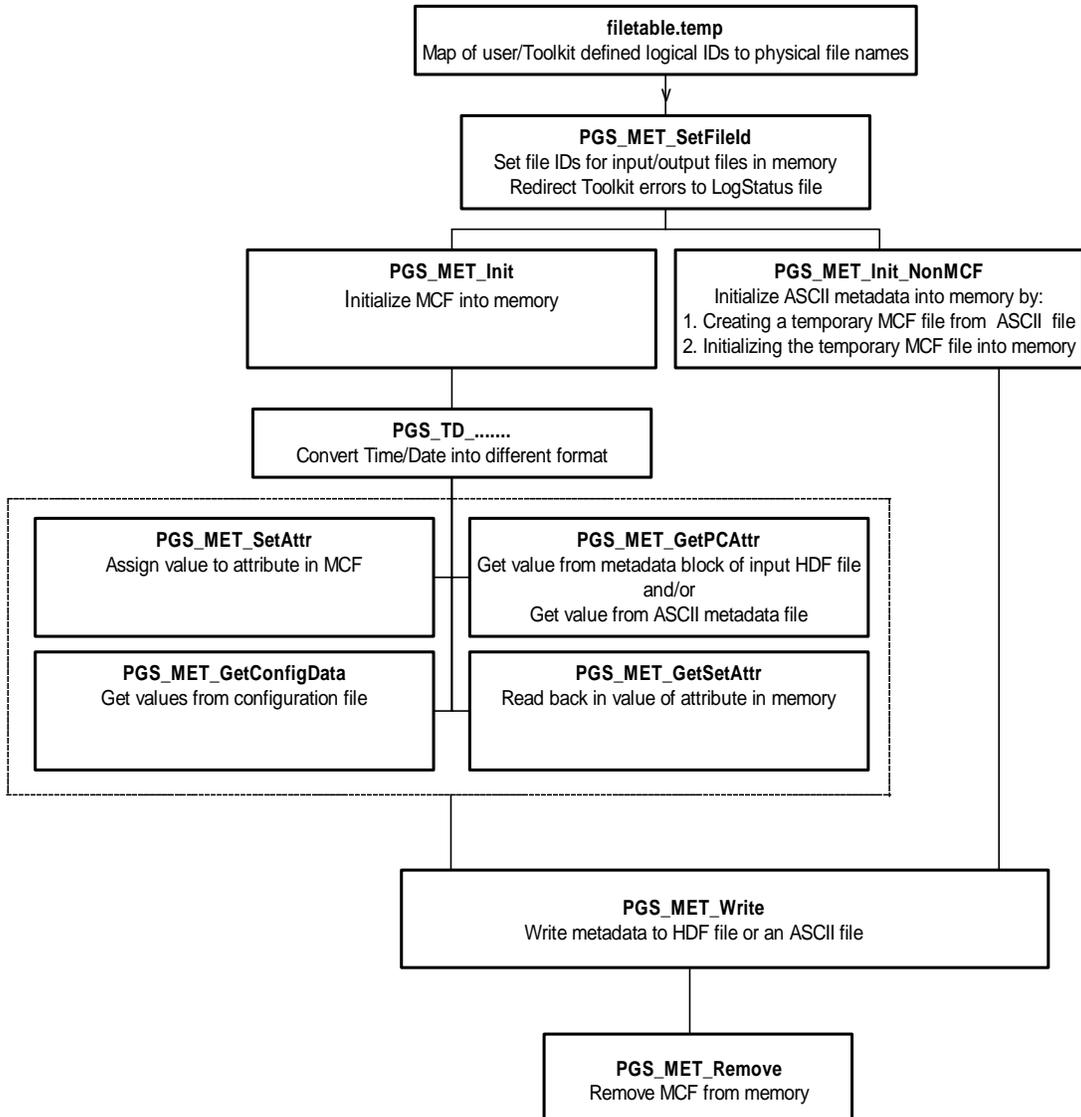


Figure 3-3. Data Flow Diagram

4. Toolkit Usage and Functionality

4.1 Introduction

The Metadata and Time/Date tools delivered with this document are extracted from the SDP Toolkit to create a stand-alone tool for the Metadata as well as the Time/Date tools. This stand-alone Toolkit, to be called Toolkit_MTD (or simply Toolkit) to distinguish it from the SDP Toolkit, functions similarly to SDP Toolkit with some differences that will be mentioned where appropriate.

4.2 Functionality

Unlike the SDP Toolkit, the Toolkit_MTD does not require Process Control Files (PCFs) as explained in the SDP Toolkit User's Guide. Instead, it uses a simple file table, referred to as Private Customized File Table (PCFT), which is explained in detail in Appendix C. This file includes all the entries for the files that are used in a run and maps logical identifiers to physical file names. After creation, this file must be copied to the directory where the executable is run. The copied file name should be **filetable.temp**. Note that some entries in this file have fixed logical identifiers and the user is only free in specifying the file name and/or the path. Other logical Ids can be any number as long as they are unique. Before calling any other MET or TD tools, the user is required to call `PGS_MET_SetFileId()` or `PGS_TD_SetFileId()`. These functions read the file identifiers and their physical names from **filetable.temp** into memory for later use by MET and TD tools. In addition, these functions redirect error messages to the LogStatus file that the user specifies in the **filetable.temp**.

In the SDP Toolkit, certain configuration parameters are held in the PCF file, and are retrieved by the `PGS_MET_GetConfigData()`, or by the PC support tools in the TD tools. Since in the Toolkit_MTD no PCF file is used, these configuration parameters are held in a "separate file such as configfile, that has an entry in the **filetable.temp** with a fixed logical identifier 5000. See Appendix F for details on this file.

4.3 MET Tools

Once the file identifiers are set, other MET tools can be used to initialize metadata files, set the attributes and write metadata to the products such as HDF files (both HDF4 and HDF5 file types). See Section 6.2.1 and 6.2.2. Examples in Appendix D shows the steps that may be taken in reading, setting, and writing metadata from input files to output files. It is worth emphasizing that in addition to initializing MCF files as in the SDP Toolkit, MET tools in the Toolkit_MTD can initialize any file in which attribute values are written as ASCII records. The initialization of the ASCII files are accomplished by a call to the `PGS_MET_Init_NonMCF` function. After the initialization, the situation is as if an MCF file has been initialized with one major difference. The values for the objects in the ASCII file are set in that file and they cannot be set using

PGS_MET_SetAttr function. (except the ProductionDateTime object which is set by the Toolkit).

The initialization of an ASCII metadata file is actually a two step process which is hidden from the user. The first step is the creation of an MCF file from the input ASCII metadata file. This is accomplished by a call to PGS_MET_ConvertToMCF inside the function PGS_MET_Init_NonMCF. The MCF file to be created must have an entry in the **filetable.temp** (see Appendix C, where this MCF file is named as a temporary MCF). The second step in the PGS_MET_Init_NonMCF is initialization of the created MCF file by a call to PGS_MCF_Init. Another added functionality to the MET tools is recovering the file Ids from their physical names. This is accomplished by a call to PGS_MET_GetFileId, giving the filename (including path) as an input. The returned fileID can then be used as an input for other MET tools that require the file ID.

Another difference with the SDP Toolkit is the location where some configuration parameters are held. In the SDP Toolkit certain configuration parameters are held in the PCF file. Those configuration parameters in the Toolkit_MTD are held in a config file which has an entry in the **filetable.temp**. Appendix F explains this file in detail. See also example 1 in Appendix D on getting the parameters stored in this file.

4.4 Time/Date Tools

Once the file identifiers are set by a call to PGS_MET_SetFileId() or PGS_TD_SetFileId() for the Time/Date tool (see Appendix A on required files for the tools), one can use other Time/Date tools to convert easily and accurately between different representations of time, such as spacecraft time, UTC, Internal Atomic Time and Julian date. After conversion, the time values can be added to metadata using MET tools, such as PGS_MET_SetAttr. The use of these tools is straightforward and ample examples are provided in section 6.2.3 and test drivers are in the directory \$PGSHOME/test/test_TIME.

4.5 Error/Status Log

Toolkit error messages are reported in the LogStatus file. The logical Id for this file in the PCFT file (i.e. **filetable.temp**) is 10100. The error redirection to this file is implemented in the PGS_MET_SetFileId or PGS_TD_SetFileId functions, since by design these are the first functions to be called by the user. If the PCFT file does not contain an entry for the LogStatus file, the error messages will be directed to the standard output. The error report mechanism is a combination of mechanisms implemented in the SDP Toolkit and HDF tools. For details refer to section 6.2.2

4.6 HDF-EOS

HDF-EOS and HDF-EOS5 are standalone packages that may be used in conjunction with the Toolkit_MTD. They implement the EOS standard methods for accessing HDF4 and HDF5 format files respectively. Three interfaces are provided: Point, Swath, and Grid. Please refer to the HDF-EOS User's Guide for more details.

5. Toolkit_MTD Installation and Maintenance

5.1 Introduction

The Toolkit_MTD can be installed on UNIX Platforms SGI (both 64-bit and new 32-bit ABIS), SUN, DEC, HP, LINUX, and IBM (please see Table 5.1 for more details). It can also be installed in PCs running Windows NT and 98. Installation procedures on UNIX Platforms are outlined in Section 5.2 and Installation Procedures for Windows NT/98 are in Section 5.3.

Note: Version 5.2.7.1 cannot be installed on Windows NT/98 since HDF5 for Windows is not available at the time of this release.

5.2 Installation Procedures for UNIX Platforms

5.2.1 Toolkit_MTD Release Notes

5.2.1.1 Multiple Architecture Support

The Toolkit_MTD has the option of being installed with simultaneous support for multiple architectures. This means that it is not necessary to install a separate copy of the Toolkit_MTD for each host architecture to be supported. Instead, a single copy of the Toolkit_MTD, installed on a file server in a networked environment, may serve multiple hosts of different architecture types.

Running concurrent tasks on the Toolkit_MTD is possible, but it requires that each process be configured so that all output files, including Toolkit_MTD log files, are written to a separate area to avoid collisions. This is done by using a Private Customized File Table (PCFT) for each concurrent task. Note that any such PCFT MUST contain required entries for proper Toolkit_MTD functioning.

The directory structure of the Toolkit_MTD allow multiple architecture support. Subdirectories of the Toolkit_MTD home directory are as follows:

bin	binary and script executables	Note 1	
database	data resource files used by the Toolkit_MTD	Note 1	
doc	documentation		
include	header files		
lib	the Toolkit_MTD library	Note 1	
obj	object files used to build the Toolkit_MTD library	Note 1	
objcpp	object files used to build C++ version of Toolkit_MTD 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_MTD environment variables are set by the environment scripts to automatically map to the appropriate directories.

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

Note 2:

The directory runtime contains sample data files shared by all architectures. Currently the only file distributed in this directory are the PCFT (filetable.template) and configfile (configfile.dat) for the test drivers in the test subdirectory.

5.2.1.2 DAAC Toolkit_MTD Support

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

5.2.1.3 Support for the IRIX 6.2 Operating System

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

<u>ABI</u>	<u>compiler flag</u>	<u>Toolkit_MTD name</u>
new-style 32 bit	-n32	sgi32
64 bit	-64	sgi64

Theses formats run only under IRIX 6.x.

5.2.1.4 HDF Integration

The Toolkit_MTD installation procedures include sections that covers the installation of the National Center for Supercomputer Applications (NCSA) 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.

HDF (i.e. both HDF4 and HDF5) is needed in order to build and use the Metadata (MET) and Date/Time (TD) tools.

An installation script for HDF is included as part of the main Toolkit_MTD distribution. It is provided to simplify the installation of HDF4 and HDF5 as much as possible, greatly reducing the number of steps in NCSA's own installation procedure. As of ECS Release 5B, the Toolkit_MTD uses HDF4.1r3 and hdf5-1.2.2-Post10. The HDF distributions themselves are

located in compressed tar files, called HDF4.1r3.tar.Z and hdf5-1.2.2-Post10.tar.Z which must be downloaded separately.

With a full installation, HDF4 and HDF5 require 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 Toolkit_MTD home directory. The disk partition where HDF4 and HDF5 are installed should have about 120 Mb of free space.

5.2.1.5 HDF-EOS Integration

The Toolkit_MTD installation procedures include a section which covers the installation of HDF-EOS, and HDF-EOS5.0 standalone packages that may be used in conjunction with the Toolkit_MTD. It implements the EOS standard methods for accessing HDF format files (Both HDF4 and HDF5). Three interfaces are provided: Point, Swath and Grid. Please refer to the HDF-EOS User's Guide for more information. As of Release B0, the Toolkit_MTD requires HDF-EOS 2.0 or later. The distribution files for HDF-EOS and HDF-EOS5 are available from the same ftp server where the Toolkit_MTD distribution files are located.

The Toolkit_MTD HDF-EOS handles the details of unpacking the distribution files, setting HDF4 and HDF5 dependencies, and running the HDF-EOS and HDF-EOS5 installation scripts.

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

5.2.2 To Install the Toolkit_MTD from a Disk-Based Tar File

5.2.2.1 Preliminary

If HDF4 and HDF5 has not been installed it should be installed at this time. You must first download the HDF4 distribution file HDF4.1r3.tar.Z and HDF5 distribution file hdf5-1.2.2-Post10.tar.Z before proceeding. They may be loaded into any directory on your system, i.e. they need not reside in the Toolkit_MTD home directory. The same applies to the HDF-EOS distribution file HDF-EOS2.7v1.00.tar.Z and HDF-EOS5 distribution file HDF-EOS5.0.tar.Z, if you plan to install HDF-EOS and HDF-EOS5 (recommended) while installing the Toolkit_MTD.

Important HDF Note:

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

Because of these factors, we strongly recommend that even if you already have HDF4.1r3 and hdf5-1.2.2-Post10 installed, you RE-INSTALL HDF4 and HDF5 AT THIS TIME, using the Toolkit_MTD-supplied HDF installation scripts.

Historical Note:

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

5.2.2.2 Unpacking the Distribution File

1. Select a location for the Toolkit_MTD 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 110 Mb of free space. If you plan to install support for multiple architectures, you will need about 20 Mb Toolkit_MTD space + 30 Mb HDF space for each additional architecture supported.

Multiple Architecture Support Note

As previously mentioned, it is now possible to build the Toolkit_MTD with support for multiple architectures (currently only SUN and SGI are supported). The distribution file need only be unpacked once, to support all architectures. If the Toolkit_MTD 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 two 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 MTDTK5.2.7.1v1.00.tar.Z to the target directory by typing the command:

```
cp MTDTK5.2.7.1v1.00.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 MTDTK5.2.7.1v1.00.tar.Z | tar xvf -
```

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

5.2.2.3 Starting the Installation Procedure

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

```
cd TOOLKIT_MTD
```

Multiple Architecture Support Note:

The Toolkit_MTD 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_MTD directory: <target-dir>/TOOLKIT_MTD. 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_MTD installation script.

Before running the Toolkit_MTD 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.

<u>Site</u>	<u>FORTRAN</u>	<u><install-options></u>
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 specifically. When installing the Toolkit_MTD in a directory which is being automounted or which is a link, the Toolkit_MTD 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_MTD directory name (e.g.: /usr/local/TOOLKIT_MTD). Note that this option can **NOT** be used to specify an installation directory other than where the TOOLKIT_MTD 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_MTD in debug mode add the following to <install-options>:

`-debug`

This will replace the optimization flag "-O" with "-g" for all files compiled into the Toolkit_MTD library. This allows Toolkit_MTD 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>`

(example: `-cpp_path /usr/bin/cpp`)

Where <C++-compiler-path> is the directory where the desired C++ compiler is located. 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>`

(example: `-cc_path /usr/bin/cc`)

Where <C-compiler-path> is the directory where the desired C compiler is located. 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>`

(example: `-fc_path /usr/bin/f90`)

Where <FORTRAN-compiler-path> is the directory where the desired FORTRAN compiler is located. This is particularly advisable when using FORTRAN-90.

NAG FORTRAN-90 Note:

If using a NAG FORTRAN-90 compiler to build the Toolkit_MTD library, add the `-nag` option to <install-options>, after the `-f90` and `-fc_path` options. This will allow the

Toolkit_MTD to generate the proper C to FORTRAN bindings. This option should not be used when building the Toolkit_MTD on an SGI. See the note, below.

SGI Multiple Architectures Note:

On the SGI (as of IRIX64 6.2), the default is to build the Toolkit_MTD in 64-bit mode. The following table gives the option to specify the appropriate architecture to be built:

<u>binary format</u>	<u>architecture</u>	<u><install-options></u>
new-style 32 bit	sgi32	-sgi32
64 bit	sgi64	-sgi64

SGI FORTRAN-90 Note:

On SGI and SGI Challenge platforms running IRIX 6.2 and earlier, the type of FORTRAN-90 compiler is automatically determined by the script. 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:

<u>binary format</u>	<u>architecture</u>	<u>f90</u>
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_MTD 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_MTD 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_MTD to successfully install without a FORTRAN HDF must be installed independently (i.e. NOT from the Toolkit_MTD INSTALL script) (see HDF Installation Section, below).

If you have already installed NCSA's HDF4 package, you can specify the installation location explicitly. If you do so, the Toolkit_MTD installation procedure will not attempt to install HDF, 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 NCSA's HDF5 package, you can specify the installation location explicitly. If you do so, the Toolkit_MTD 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 package, you can specify the installation location explicitly. If you do so the Toolkit_MTD 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 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 package, you can specify the installation location explicitly. If you do so the Toolkit_MTD 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 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 appropriate version level for the version of the Toolkit_MTD being installed.

By default the Toolkit_MTD 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 HDF and/or HDF-EOS, these packages will be installed under the TOOLKIT_MTD 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 `-debug` switch the Toolkit_MTD, HDF4, HDF5, HDF-EOS and HDF-EOS5 will all be installed in debug mode. Finally if you attempt to install HDF4 (or HDF5) and an installed HDF4 (or HDF5) is found in the default location it will be deleted and the whole HDF (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".

5.2.2.4 Run the Toolkit_MTD Installation Script

Please note that the installation script for this release of the Toolkit_MTD 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 five other scripts: bin/INSTALL-HDF4, bin/INSTALL-HDF5, bin/INSTALL-HDFEOS-Wrap, bin/INSTALL-HDFEOS5-Wrap, 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_MTD Installation starting at <date/time>
```

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.

HDF4 Installation Section

1. The script prompts with:

```
Is HDF4.1r3 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 HDF4.1r3 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_MTD Installation Section, below.

3. The script prompts with:

```
Do you wish to install HDF4.1r3 now [yes] ?
```

Hit return to continue.

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 HDF4.1r3.tar.Z is located ?

Please enter the correct location and hit return.

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

Pathname where directory 'HDF4.1r3' 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_MTD installation. We therefore recommend using the default HDF directory, suggested by the installation procedure, as it helps keep track of which architecture was used to build HDF.

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 HDF installation. When the HDF section is complete, it outputs the message:

HDF installation ending at: <date/time>

Note regarding HDF on DEC Digital Unix: Please see Warning in Section 5.6.

HDF5 Installation Section

1. The script prompts with:

Is hdf5-1.2.2-Post10 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 hdf5-1.2.2-Post10 directory is located [<default>] ?

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

3. The script prompts with:

Do you wish to install hdf5-1.2.2-Post10 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 hdf5-1.2.2-Post10.tar.Z 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_MTD-home-directory>/hdf5/\$BRAND, where \$BRAND is the Toolkit_MTD architecture being built, given by the table in Note 2 of the NOTES section, below.

Pathname where directory ' hdf5-1.2.2-Post10' 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_MTD installation. We therefore recommend using the default HDF 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-EOS2.7v1.00 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-EOS2.7v1.00 is installed [<default-path>]

3. The script prompts with:

Do you wish to install HDF-EOS2.7v1.00 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-EOS2.7v1.00.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_MTD-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 hdfs 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-E5.0 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-EOS5.0 is installed [<default-path>]

3. The script prompts with:

Do you wish to install HDF-EOS5.0 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-EOS5.0.tar.Z 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_MTD-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-EOS5, Refer to the file README in the HDF-EOS5 'doc' directory.

Toolkit_MTD Installation Section

1A. SCF Installation

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

Running the TOOLKIT_MTD Installation Script . . .

TOOLKIT_MTD installation script: INSTALL-Toolkit

Starting at: <date/time>

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

1B. DAAC Installation

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

Running the TOOLKIT_MTD Installation Script ...

TOOLKIT_MTD installation script: INSTALL-Toolkit

Starting at: <date/time>

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

1C. C++ Installation

If the C++ version of the Toolkit_MTD is being built (-cpp option) 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.

4. The Toolkit_MTD 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.

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

TOOLKIT_MTD installation completed at <date/time>

Otherwise messages of the following form will appear:

INSTALL: Error: <error message>

TOOLKIT_MTD 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.

5.2.2.5 User Account Setup

Once the Toolkit_MTD has been installed, the accounts of Toolkit_MTD users must be set up to define environment variables needed to compile and run code with the Toolkit_MTD (see parts 2 and 3 of the Notes section 5.2.2.8, below). The type of setup depends on the user's login shell.

- 1A. C shell (csh) users:

Edit the Toolkit_MTD 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_MTD 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_MTD bin directory to the user path.

The script `pgs-dev-env.csh` sets up all of the variables set by `pgs-env.csh` and adds the `Toolkit_MTD` 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 `Toolkit_MTD` 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>/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 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 makes 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 `Toolkit_MTD` 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 Toolkit_MTD 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_MTD 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_MTD 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_MTD 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 regarding path setup with pgs-dev-env.ksh and pgs-dev-env.ksh.cpp:

The scripts pgs-dev-env.ksh and pgs-dev-env.ksh.cpp also makes 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 Toolkit_MTD 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_MTD 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 Toolkit_MTD user's .profile, where <SDP-home-dir> is the full path of the Toolkit_MTD 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.

5.2.2.6 File Cleanup

Once the Toolkit_MTD 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
```

5.2.2.7 Rebuilding the Toolkit_MTD library

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

SCF Installation

To rebuild the Toolkit_MTD 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 of 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.

5.2.2.8 NOTES:

4. The Toolkit_MTD was built and tested in a multi-platform environment using the following platforms, operating systems, and compilers:

Table 5-1. SDP Toolkit_MTD Development Configuration

Platform	OS	Version	C Compiler	FORTRAN	C++ Compiler
SGI	IRIX	6.2	SGI C 7.1	SGI FORTRAN 7.1	SGI C++
Sun	Solaris	2.5.1	Sun C 4.0	Sun FORTRAN 4.0	SUN C++ 4.1
DEC	Digital Unix	4.0	Dec C 4.10	DEC FORTRAN 4.10	
HP	HP-UX	10.01	HPC 10.24	HP FORTRAN 10.24	
HP11	HP-UX11	11.0	A.11.01.03 HPC	HP F90 V2.3	
IBM	AIX	4.2	IBMC 3.1.4	IBM FORTRAN 3.2.5	
LINUX	Linux	2.2.16-22	GNU gcc	GNU g77	

Notes:

- a. SGI is 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_MTD architecture type names
 To track architecture dependencies, the Toolkit_MTD defines the environment variable \$BRAND. Following is a list of valid values for this variable, which is referred to throughout this document:

<u>\$BRAND</u>	<u>Architecture</u>
dec	DEC Alpha
hp	HP 9000, HP-UX11
ibm	IBM RS-6000
sgi	SGI (old-style 32-bit ABI)
sgi32	SGI (new-style 32-bit ABI)
sgi64	SGI (64-bitABI)
sun5	Sun: SunOS 5.x (Solaris 2.x)
linux	Linux

3. In order to use the Toolkit_MTD 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_MTD (where <install-path> is the absolute directory path above Toolkit_MTD)	top level directory
PGSBIN	\${PGSHOME}/bin/(\$BRAND)	executable files
PGSDAT	\${PGSHOME}/database/ (\$BRAND)	Toolkit_MTD database files
PGSINC	\${PGSHOME}/include	include (header) files
PGSLIB	\${PGSHOME}/lib/(\$BRAND)	library files
PGSOBJ	\${PGSHOME}/obj/(\$BRAND)	Toolkit_MTD object files
PGSRUN	\${PGSHOME}/runtime	runtime work files
PGSSRC	\${PGSHOME}/src	Toolkit_MTD source files
PGSTST	\${PGSHOME}/test	test area
PGSCPPO	\${PGSHOME}/objcpp/(\$BRAND)	Toolkit_MTD C++ object files

4. Other Toolkit_MTD 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
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
CPPFLAGS	Default C++ Flags
CPPFHFLAGS	CPPFLAGS
CPP	C++ Compiler

- For a complete list of the tools provided with this release of the Toolkit_MTD, please refer to Section 1, Table 1–2
- The majority of the Toolkit_MTD functions are written in C. These C–based tools include the file cfortran.h, using it to generate machine–independent FORTRAN bindings.

5.2.3 Compiling User Code with the Toolkit_MTD

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

The Toolkit_MTD includes command files that set up environment variables to simplify the task of compiling with Toolkit_MTD 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)
F77	the name of the FORTRAN compiler (usually f77)
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_MTD include files
PGSLIB	the location of the Toolkit_MTD library libPGSTK.a
HDFINC	HDF include files
HDFLIB	HDF Library files

CPP The name of the C++ Compiler (usually CC)

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

for csh users, type:

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

for ksh users, type:

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

where <Toolkit_MTD-HOME-DIRECTORY> is the location where the Toolkit_MTD 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.

Since the Toolkit_MTD was built with HDF support, you may use the lines listed below for compiling with HDF support:

C to object:

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

C++ to object:

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

C to executable:

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

C++ to object:

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

FORTTRAN to object:
\$F77 -c \$F77_CFH myfile.f

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

The important thing in this case is that your code gets linked with the HDF library. You do not need -\$HDFINC unless your C or C++ code makes direct calls to HDF.

5.3 Installation Procedures for Windows NT/98

The instructions which follow assume that you will be using one of the 'zip' files that we provide, either the source code release with fortran (pgstk5.2.7r1full.zip) or the source release with no fortran(pgstk5.2.7r1nof.zip).

In building TOOLKIT_MTD from source code you may select between two build environment options depending on your application and environment needs. Each option has it's own zip file:

Option I, (select pgstk5.2.7r1nof.zip):

Test and Utility configuration : TOLKIT_MTD library, and utilities, no fortran

Option II, (select pgstk5.2.7r1full.zip):

Full configuration : TOOLKIT_MTD library and utilities, with fortran. This version has been built and tested using DEC Visual Fortran.

Preconditions:

To build the TOOLKIT_MTD library, it is assumed that you have done the following:

1. Installed MicroSoft Developer Studio and Visual C++ 6.0. Visual Fortran 6.0 is needed if you are going to build the full TOOLKIT_MTD Library with Fortran support.
2. Installed NSCA HDF version4.1 release 3.
3. Set up a directory structure to unpack the library. For example:

c:\ (any drive)
PGSTKHOME\ (any folder name)

4. Copy the source distribution archive to that directory and unpack it using the appropriate archiver options to create a directory hierarchy.

INSTRUCTIONS FOR LIBRARY AND UTILITY INSTALLATION, NO FORTRAN :

1. You will use pgstk5.2.7r1nof.zip

Run WinZip on c:\PGSTKHOME\pgstk5.2.7r1nof.zip. This should create directories called 'dev' and 'TOOLKIT_MTD'. The 'dev' directory contains a Developer Studio workspace TOOLKIT_MTD.dsw.

2. Invoke Microsoft Visual C++ 6.0, go to "File" and select "Open Workspace" option. Then open c:\PGSTKHOME\dev\TOOLKIT_MTD.dsw workspace.

3. Under Tools->Options, select the folder, Directories; Under "Show directories for", select "Include files". Add the following directories:

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE

C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE

C:<path to HDF includes>\INCLUDE

C:\PGSTKHOME\TOOLKIT_MTD\INCLUDE

C:\PGSTKHOME\TOOLKIT_MTD\INCLUDE\CUC

C:\PGSTKHOME\dev

4. Select "Build", then Select "Set Active Configuration". Select "PGSTK -- Win32Debug" as active configuration. Select "Build" and "Build PGSTKd.lib" to build the Debug version of the TOOLKIT_MTD tree.

5. Select "Build", then Select "Set Active Configuration". Select "PGSTK -- Win32Release" as active configuration. Select "Build" and "Build PGSTK.lib" to Build the Release version of the TOOLKIT_MTD tree.

6. Under Tools->Options, select the folder, Directories;

Under "Show directories for", select "Library files". Add the following directories:

C:\Program Files\Microsoft Visual Studio\VC98\LIB

C:\Program Files\Microsoft Visual Studio\VC98\MFC\LIB

C:<path to HDF library>\LIB

C:\PGSTKHOME\dev\PGSTK\Release

Select "Build", then Select "Set Active Configuration".

Select "PGS_TD_NewLeap -- Win32Release" as active configuration.

Select "Build" and "Build PGS_TD_NewLeap.exe" to build the Release version of the TOOLKIT_MTD utility PGS_TD_NewLeap and eptobin.

7. Run the installation batch file win32ins.bat in directory C:\PGSTKHOME\dev\.

Commands in this file will copy pgstk.lib, pgstk.d.lib and utilities to

C:\PGSTKHOME\TOOLKIT_MTD\bin\nt_98 and

C:\PGSTKHOME\TOOLKIT_MTD\lib\nt_98, and remove the temporary files.

The C:\PGSTKHOME\TOOLKIT_MTD\bin\nt_98 directory will contain utilities:

PGS_TD_NewLeap.exe

ephtobin.exe

The C:\PGSTKHOME\TOOLKIT_MTD\lib\nt_98 directory will contain PGSTK libraries:

pgstk.lib (TOOLKIT_MTD Library of release version)

pgstk.d.lib (TOOLKIT_MTD library of debug version)

INSTRUCTIONS FOR FULL TOOLKIT_MTD INSTALLATION WITH FORTRAN:

1. You will use pgstk5.2.7r1full.zip

Run WinZip on c:\PGSTKHOME\pgstk5.2.7r1full.zip.

This should create directories called 'dev' and 'TOOLKIT_MTD'. The dev' directory contains a Developer Studio workspace TOOLKIT_MTD.dsw.

2. Invoke Microsoft Visual C++ 6.0, go to "File" and select "Open Workspace" option. Then open c:\PGSTKHOME\dev\TOOLKIT_MTD.dsw workspace.

3. Under Tools->Options, select the folder, Directories; Under "Show directories for", select "Include files". Add the following directories:

C:\Program Files\Microsoft Visual Studio\DF98\IMSL\INCLUDE

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE

C:\Program Files\Microsoft Visual Studio\DF98\INCLUDE

C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE

C:<path to HDF includes>\INCLUDE

C:\PGSTKHOME\TOOLKIT_MTD\INCLUDE

C:\PGSTKHOME\TOOLKIT_MTD\INCLUDE\CUC

C:\PGSTKHOME\dev

4. Select "Build", then Select "Set Active Configuration".

Select "PGSTK -- Win32Debug" as active configuration.

Select "Build" and "Build PGSTKd.lib" to build the Debug version of the TOOLKIT_MTD tree.

5. Select "Build", then Select "Set Active Configuration".

Select "PGSTK -- Win32Release" as active configuration.

Select "Build" and "Build PGSTK.lib" to build the Release version of the TOOLKIT_MTD tree.

6. Under Tools->Options, select the folder, Directories;

Under "Show directories for", select "Library files". Add the following directories:

C:\Program Files\Microsoft Visual Studio\DF98\LIB

C:\Program Files\Microsoft Visual Studio\DF98\IMSL\LIB

C:\Program Files\Microsoft Visual Studio\VC98\LIB

C:\Program Files\Microsoft Visual Studio\VC98\MFC\LIB

C:<path to HDF library>\LIB

C:\PGSTKHOME\dev\PGSTK\Release

Select "Build", then Select "Set Active Configuration".

Select "PGS_TD_NewLeap -- Win32Release" as active configuration.

Select "Build" and "Build PGS_TD_NewLeap.exe" to build the Release version of the TOOLKIT_MTD utility.

7. Run the installation batch file win32ins.bat in directory C:\PGSTKHOME\dev\.

Commands in this file will copy pgstk.lib, pgstk.d.lib and utilities to

C:\PGSTKHOME\TOOLKIT_MTD\bin\nt_98 and

C:\PGSTKHOME\TOOLKIT_MTD\lib\nt_98, and remove the temporary files.

The C:\PGSTKHOME\TOOLKIT_MTD\bin\nt_98 directory will contain utilities;

PGS_TD_NewLeap.exe

ephtobin.exe

The C:\PGSTKHOME\TOOLKIT_MTD\lib\nt_98 directory will contain PGSTK libraries:

pgstk.lib (TOOLKIT_MTD Library of release version)

pgstk.d.lib (TOOLKIT_MTD library of debug version)

Compile Notes:

If you are building an application that uses the TOOLKIT_MTD library pgstk.lib (release version) or pgstk.d.lib (debug version), the following locations will need to be specified for locating header files and linking in the HDF and TOOLKIT_MTD libraries:

<top-level HDF directory>\lib

<top-level PGSTK directory>\TOOLKIT_MTD\lib\nt_98

<top-level HDF directory>\include

<top-level PGSTK directory>\TOOLKIT_MTD\include

<top-level PGSTK directory>\TOOLKIT_MTD\include\CUC

<top-level PGSTK directory>\dev

where <top-level PGSTK directory> may be C:\PGSTKHOME

Please refer to the <top-level PGSTK directory>\dev\compile_NT_98.txt file for more information on compiling an application with the TOOLKIT_MTD libraries on Windows 98 and Windows NT.

In order to use PGSTK.lib or PGSTKd.lib, it is needed to set up environment variable "PGSHOME" :

Open autoexe.bat file and add 1 line at the end of file:

Set PGSHOME=<top-level PGSTK directory>\TOOLKIT_MTD where <top-level PGSTK directory> may be C:\PGSTKHOME

MORE HELPFUL POINTERS:

Here are some notes that may be of help is you are not familiar with using the Visual C++ Development Environment.

Project name and location issues:

The files in pgstk5.2.7r1full.zip (or pgstk5.2.7r1nof.zip) must end up in the dev\ and TOOLKIT_MTD\ directories installed by pgstk5.2.7r1full.zip (or pgstk5.2.7r1nof.zip).

If you must install TOOLKIT_MTD.dsw in another directory, relative to dev\ , you will be asked to locate the project files, when you open the project TOOLKIT_MTD.dsw.

Settings... details:

If you create your own project, the necessary settings can be read from the TOOLKIT_MTD.dsw file(as text), or from the Project Settings in the Developer Studio project settings dialog.

Project

Settings

C/C++

Category

PreProcessor

Code Generation

Use run-time Library

These are all set to use Single-Threaded.

or Single-Threaded debug.

5.4 Instructions on Making Changes to Installation Procedures for UNIX Platforms

The installation procedures given in the subsection 5.2 should work seamlessly for a platform in Table 5–1. This subsection gives instructions on making changes to the installation procedure of subsection 5.2, 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 Toolkit_MTD home directory.

- a. After unpacking the tar file, but before running bin/INSTALL, (steps a–e in Section 5.2, edit the file INSTALL in <SDP-home-dir>/bin.

The section starting with the comment at line #176 and ending at line 330 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	Sun Sparc (SunOS 5.X)
Sgi	SGI Indigo
Hp	HP 9000
Dec	DEC Alpha
Ibm	IBM RS-6000
linux	Linux

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
CPP	C++ compiler
CPPFLAGS	Default C++ flags
CPPFHFLAGS	CPPFLAGS + CPP_CFH
CFHFLAGS	CFLAGS + C_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++
C_CFH	needed to compile C Toolkit_MTD code that uses cfortran.h for FORTRAN bindings
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 Toolkit_MTD.

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

The section starting with comment at line #84 and ending at line #185 is identical to the previously mentioned section in the file bin/INSTALL, and must be modified in the same way.

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

5.5 Link Instructions

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

The delivery consists of a single Toolkit_MTD library called libPGSTK.a.

Here we give generic command lines for linking with this library. We use \$C_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_MTD, 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_MTD, on all machines:

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

NOTES:

Specific examples on how to link particular Toolkit_MTD functions on the Toolkit_MTD development platforms are given with the supplied tool test drivers. See the "Test Drivers" in Section 5.6.

If you are using a different development configuration than one of those given in table 5-1 ("Toolkit_MTD Development Configuration") of Section 5.2, see Section 5.4 ("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 Toolkit_MTD where possible. These switches enforce ANSI/POSIX standards, necessary for compiling the Toolkit_MTD 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 `$PGSHOME/bin/pgs_dev_env.csh`. `$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 Toolkit_MTD with the C and FORTRAN compilers respectively.

5.6 Test Drivers

Also included with this Toolkit_MTD delivery is test driver programs located in the test subdirectory.

These test programs are provided to aid the user in the development of software using the Toolkit_MTD. The user may run the same test cases as included in this file to verify that the Toolkit_MTD is functioning correctly. These programs were written to support the internal test of the Toolkit_MTD and are not an official part of the Toolkit_MTD delivery; users make use of them at their own risk. No support will be provided to the user of these programs. The test directory 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.

Warning for DEC Digital Unix Platform Users:

The FORTRAN testdrivers for MET tools may fail to create executables for the drivers. This is due to the fact that the early release of HDF4.1r3 did not support FORTRAN on DEC, therefore, the library `libmfhdf.a` does not contain FORTRAN object files. NCSA is aware of this problem on the DEC platform, and will fix it in their next release of HDF. This problem only affects the DEC platform. With version 5.2.7.1 of Toolkit_MTD a fix was made in `INSTALL-HDF4.1r3` script to solve the problem. If problem still exist (which is unlikely) follow these steps to correct the problem:

1. Go to the directory “<TOOLKIT_MTD-home-directory>/bin/dec” and type:
source pgs-dev-env.csh
2. Go to the directory where HDF4.1r3 is installed. Go to the subdirectory mfhdf/fortran and edit “Makefile”. Replace “FC = NONE” with “FC = f77”. Terminate edit and type:
make all
3. Go to the directory mfhdf/libsrc and type:
cp libmfhdf.a ../../lib/
to replace libmfhdf.a that already exists in the HDF4.1r3/lib directory.
4. Try compiling fortran testdrivers that failed earlier, by following the steps in the README_MET file.

5.7 User Feedback Mechanism

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

- a. An account at the ECS Landover facility has been set up for user response:
pgstlkit@eos.hitc.com
- b. Users will e-mail problem reports and comments to the above account. A receipt will be returned to the sender. A workoff 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_MTD will be placed in a recommended requirements data base (RRDB) and handled more formally.
- c. The following format will be used for email response.
 - 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_MTD 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. In addition to the email response mechanism, a phone answering machine is also provided. The telephone number is: 301-925-0781. Calls will be returned as soon as possible. We note that the email user response mechanism has been in operation for several years and has been effective in gathering user feedback. Email is our preferred method of responding to users.

This page intentionally left blank.

6. Toolkit Specification

6.1 Introduction

In this section, we give a descriptive list of Toolkit software tools designed to satisfy the requirements for metadata and time tools listed in *PGS Toolkit Requirements Specification for the ECS Project*, Hughes Information Technology Systems, Inc. 193-801-SD4-001, October 1993 and updated in versions through May 2000. 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 metadata formatting and time/date requests must be made through the Toolkit, as explained in section 4.1. These tools are described in Section 6.2.

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, TD=time and date conversion, PC=ProcessControl, CBP=Celestial Body Position, 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 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 Toolkit Tools

6.2.1 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 D of this document.

The metadata tools in the 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).

The first tool to be called is `PGS_MET_SetFileId`. This function sets logical file ID for the files that are used by MET tools. The tool reads logical Ids assigned for the MCF, ASCII, configfile, and other temporary input/output files in the PCFT file called **filetable.temp**, that should reside in the directory where the executable is run. A template for this file is given in Appendix C. The defined logical IDs can be used as input for other MET tools. These IDs for the file listed in the PCFT can also be recovered using `PGS_MET_GetFileId`.

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**. 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. Note that in order to write metadata to an HDF file user needs to open HDF file before calling **PGS_MET_Write**. If the HDF file is of type HDF4 user may still call HDF’s **SDstart** for this purpose. 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.

In addition to MCF, attribute values can be written into any file as ASCII records. This kind of file should be initialized by **PGS_MET_Init_NonMCF**. This routine runs the file against the parser and creates a temporary MCF file. The temporary MCF file is then initialized by **PGS_MET_Init** as an ordinary MCF file.

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 config file that is identified in the PCFT file.

The FORTRAN versions of **PGS_MET_SetAttr**, **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 D, 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:

Establish Logical IDs for Files to be Used

NAME: PGS_MET_SetFileId()

SYNOPSIS:

C: #include <PGS_MET.h>

PGSt_SMF_status
PGS_MET_SetFileId()

FORTRAN: include 'PGS_SMF.h'
include 'PGS_tk.f'

integer function pgs_met_setfileid()

DESCRIPTION: This tool sets logical IDs assigned for the user defined files in PCFT file filetable.temp.

INPUTS: None

OUTPUTS: None

RETURNS:

Table 6-1. PGS_MET_SetFileId Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSPC_E_FILE_OPEN_ERR	Error opening file File_Table or LogStatus file

EXAMPLES:

C:

```

PGSt_SMF_status  returnstatus;
returnstatus = PGS_MET_SetFileId();
if (returnstatus != PGS_S_SUCCESS)
{
    *** do some error handling ***
    :
    :
}
    
```

FORTTRAN:

```
implicit none
integer returnstatus
integer PGS_MET_SetFileId
returnstatus = PGS_MET_SetFileId()
if (returnstatus .ne. PGS_S_SUCCESS) goto 999
```

NOTES: None

Get Logical ID for a File listed in the PCFT file

NAME: PGS_MET_GetFileId()

SYNOPSIS:

C: #include <PGS_MET.h>

PGSt_SMF_status
PGS_MET_GetFileId(
char *filename)

FORTRAN: include 'PGS_SMF.h'
include 'PGS_tk.f'

integer function pgs_met_getfileidf(filename)

character*(*) filename

DESCRIPTION: This tool retrieves logical ID assigned for a file entry in PCFT file filetable.temp. It returns FileId if successful, 0 otherwise.

INPUTS:

Table 6-2. PGS_MET_GetFileId Inputs

Name	Description	Units	Min	Max
filename	File name (with full path) for an entry in PCFT	none	variable	variable

OUTPUTS: None

RETURNS:

Table 6-3. PGS_MET_GetFileId Returns

Return	Description
File ID	Successful return
0 (zero)	Failed to find ID

EXAMPLES:

C:

```
PGSt_SMF_status fileid;  
fileid = PGS_MET_GetFileId("/home/username/MY_MCF_file");
```

```
if (fileid == 0)
{
*** do some error handling ***
:
:
}
```

FORTTRAN:

```
implicit none
integer fileid
integer PGS_MET_GetFileIdF
fileid = PGS_MET_GetFileIdF("/home/username/MY_MCF_file")
      if(fileid .eq. 0) goto 999
```

NOTES: None

Open HDF File of Type HDF4 or HDF5 for Writing Metadata

NAME: PGS_MET_SDstart()

SYNOPSIS:

```
C:      #include <PGS_MET.h>
        #include <PGS_tk.h>

        PGSt_SMF_status
        PGS_MET_SDstart(
        char   *filename,
        uintn  access_mode,
        PGSt_integer *HDFfid)
```

```
FORTRAN: include 'PGS_SMF.f'
          include 'PGS_tk.f'

          integer function pgs_met_sfstart(filename, access_mode, hdfid)

          character*(*) filename
          integer hdfid
```

DESCRIPTION: This tool opens the HDF files of type HDF4 and/or HDF5 and initializes the SD interface.

INPUTS:

Table 6-4. 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 file. It can be: DFACC_RDONLY, DFACC_RDRW, DFACC_CREATE for HDF4 files and H5F_ACC_RDONLY, H5F_ACC_RDWR for HDF5 files	none		

OUTPUTS:

Table 6-5. PGS_MET_SDstart Outputs

Name	Description	Units	Min	Max
HDFfid	SD id of the file opened	none	N/A	N/A

RETURNS:**Table 6-6. 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",
                             H5F_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",
*                               H5F_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>
 #include <PGS_tk.h>

 PGSt_SMF_status
 PGS_MET_SDend(
 cha PGSt_integer HDFfid)

FORTRAN: include 'PGS_SMF.f'
 include 'PGS_tk.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 6-7. 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 6-8. 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

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.f"
          include "PGS_SMF.f"
          include "PGS_tk.f"

          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 6-9. PGS_MET_Init Inputs

Name	Description	Units	Min	Max
fileId	MCF file id	none	variable	variable

OUTPUTS:

Table 6-10. PGS_MET_Init Outputs

Name	Description	Units	Min	Max
mdHandles	metadata groups in MCF	none	N/A	N/A

RETURNS:

Table 6-11. 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 PCFT file "filetable.tmp"
10253|hdfctestfile|/home/asiyyid/pgetest/fortran/hdfctestfile
: */

#define ODL_IN_MEMORY 0
int main()
{
PGSt_MET_all_handles handles;
char * fileName = "/home/modis/hdfctestfile"; /* the
user should change this accordingly */
int32 hdfRet, sdid;
extern AGGREGATE PGSt_MET_MasterNode;
PGSt_SMF_status ret = PGS_S_SUCCESS;
PGSt_integer fileId = PGSt_MET_MCF_FILE;
PGSt_integer i;
double dval, dval[6];
char* sval;
sval = (char*) malloc(30);
ret= PGS_MET_SetFileId();
```

```

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_tk.f "
include "PGS_MET.f"
C   the file id must also be defined in the PCFT file filetable.temp as
follows
C   10253|hdfstestfile|/home/asiyyid/pgetest/fortran/hdf
C   testfile
        integer pgs_met_init
        integer pgs_met_SetFileId
        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 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/hdfstestfile"
        result = pgs_met_setfileid()

```

```
        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 D.

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

Initialize an ASCII Metadata File into Memory

NAME: PGS_MET_Init_NonMCF()

SYNOPSIS:

```
C:      #include "PGS_MET.h"

        PGSt_SMF_status
        PGS_MET_Init_NonMCF(
            PGSt_PC_Logical      fileId,
            PGSt_MET_all_handles mdHandles)
```

```
FORTRAN: include "PGS_tk.f"
          include "PGS_SMF.f"

          integer function pgs_met_init_nonmcf(fileId, mdHandles)

          integer      fileId
          character*   PGS_MET_GROUP_NAME_L
                    mdHandles(PGS_MET_NUM_OF_GROUPS)
```

DESCRIPTION: Initializes an ASCII file containing metadata.

INPUTS:

Table 6-12. PGS_MET_Init_NonMCF Inputs

Name	Description	Units	Min	Max
fileId	ASCII file id	none	variable	variable

OUTPUTS:

Table 6-13. PGS_MET_Init_NonMCF Outputs

Name	Description	Units	Min	Max
mdHandles	metadata groups in temporary MCF file created	none	N/A	N/A

RETURNS:

Table 6-14. PGS_MET_Init_NonMCF Returns

Return	Description
PGSMET_E_PCREAD_ERR	Unable to obtain filename or attribute filename from the PC Table
PGSMET_E_ODL_MEM_ALLOC	ODL routine failed to allocate memory
PGSMET_E_SD_START	Unable to open the HDF file
PGSMET_E_OPEN_ERR	Unable to open temporary input file with file ID <aggregate name>
PGSMET_E_FINDATTR	Unable to get the attr index
PGSMET_E_SD_INFO	Unable to retrieve SD attribute information
PGSMET_E_MALLOC_ERR	Unable to allocate memory for the HDF attribute
PGSMET_E_SD_READ	Unable to read HDF attribute
PGSMET_E_ODL_READ_ERR	Unable to create ODL tree <aggName> with file ID <FileId>
PGSMET_E_TYPE_ERR	Unable to obtain data type for the unset attribute
PGSMET_E_CONVERT_ERR	Unable to convert HDF-EOS metadata product file, in which unset attributes were defined as NOT SET for Data Location PGE, NOT SUPPLIED for Data Location MCF, and NOT FOUND for Data Location NONE, to an MCF file
PGSMET_E_OPEN_ERR	Could not produce temporary MCF file for metadata file
PGSMET_E_NO_Initialization	Could not initialize temporary MCF file produced for the metadata file

EXAMPLES:

```
C:      #include "PGS_MET.h"
        #define      INVENTORYMETADATA 1
        #define      MODIS_FILE 10253 /* This value must also be defined in
        the PCFT file "filetable.tmp
           10253|hdfctestfile|/home/asiyyid/pgetest/fortran/hdfctestfile
        :      */

        #define      ODL_IN_MEMORY 0
        int main()
        {
        PGSt_MET_all_handles handles;
        char * fileName = "/home/modis/hdfctestfile"; /* the
           user should change this accordingly */
        int32 hdfRet, sdid;
        extern AGGREGATE PGSg_MET_MasterNode;
        PGSt_SMF_status ret = PGS_S_SUCCESS;
        PGSt_integer fileId = "user defined ASCII file ID";
        PGSt_integer i;
        double dval, dval[6];
        char* sval;
        sval = (char*) malloc(30);
        ret=PGS_MET_SetFileId();
```

```

ret= PGS_MET_Init_NonMCF(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_tk.f"
include "PGS_MET.f"
C   the file id must also be defined in the PCFT file filetable.temp as
follows
C   10253|hdfstestfile|/home/asiyyid/pgetest/fortran/hdf
C   testfile
        integer pgs_met_init_nonmcf
        integer pgs_met_SetFileId
        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  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/hdfstestfile"
        result = pgs_met_setfileid()
                result = pgs_met_init_nonmcf("user defined ASCII file ID",

```

groups)

```
if(result.NE.PGS_S_SUCCESS) then
print *, "Initialization error. See Logstatus for details"
endif
print *, "SUCCESS"
end
```

NOTES: None

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_tk.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 6-15. 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 6-16. 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 =
1    pgs_met_setattr_d(groups(INVENTORYMETADATA),
    "GRingPointLatitude.1", dvals)

C For setting Product Specific Attributes

    informationname = "TestingAttribute1"
1    ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
    "AdditionalAttributeName.1", informationname)
    informationname = "testingAttributeValue1"
1    ret = pgs_met_setattr_s(groups(INVENTORYMETADATA),
    "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 D 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 D.

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", "MCF", "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:

```
.  
.br/>char a[100];  
.br/>.br/>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"  
.br/>.br/>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

```
.br/>.br/>char a[10][100];  
.br/>.br/>strcpy(a[0], "MODIS");  
retVal = PGS_MET_SetAttr(mdHandles[GROUP_GRANULE_DATA],  
"SATELLITE_NAME", &a[0]);
```

while the following is acceptable:

```
.br/>.br/>char *a[10];  
.br/>.br/>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

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_tk.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 6-17. 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 6-18. 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 6-19. 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_tk.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 6-20. 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 6-21. 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 6-22. 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    " TestingAttributel",&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_getpccattr_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 PCFT:

```
10253|hdfinputfile|/my/product/directory/hdfinputfile
```

The file names in the second and third fields must be identical. If the input file is not in HDF, the metadata will be read from an ASCII file which must be separately identified in the third field of the input product entry of the PCFT, as shown in this example:

```
10253|inputfile|/my/product/directory/inputfile.met
```

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 Config File Table

NAME: PGS_MET_GetConfigData()

SYNOPSIS:

C: #include "PGS_MET.h"

```
PGSt_SMF_status
PGS_MET_GetConfigData(
    char*      attrName,
    void*      attrValue)
```

FORTRAN: include "PGS_tk.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 configfile table as follows

10220|REMOTEHOST|sandcrab

This tool would retrieve the value "sandcrab" from the configfile 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 6-23. PGS_MET_GetConfigData Inputs

Name	Description	Units	Min	Max
attrName	name of parameter in configfile	none	N/A	N/A

OUTPUTS:

Table 6-24. 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 6-25. 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 configfile". Likelihood is that either the parameter does not exist in the configfile or the configfile itself is in error.
PGSMET_E_CONFIG_CONV_ERR	"Unable to convert the value of configuration parameter <name> from the configfile 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);
```

FORTTRAN:

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 configfile depicting the config id, it is still important that this field is unique in the configfile 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 configfile 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_tk.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 6-26. 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 6-27. 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
```

```

1      result= pgs_met_write(groups(INVENTORYMETADATA),
      "coremetadata", sdid)
      if(result.NE.PGS_S_SUCCESS.AND.
1      result.NE.PGSMET_MAND_NOT_SET) then
      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.)

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 PCFT in order to create this required .met file:

```

100 | ProductName | my/output/directory/productName
.
.
.

```

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 PCFT (i.e. 100).

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:

CALENDARDATE 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.

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_tk.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: PGSTK-0430

6.2.2 Error/Status Reporting (SMF Tools)

To detect and report on 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 similar to the one used by HDF and HDF-EOS. The error codes are enumerated in such a fashion that each code corresponds to a status identifier. These identifiers take the form of defined mnemonics that visually conveys the essence of the status message.

Thus the 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 as mentioned above. 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).

The error reporting tools report errors in a file that is identified by the logical ID of 10100. An entry such as

```
10100|LogStatus|<path>/LogStatus
```

in the PCFT file will direct errors to the LogStatus file. This file will be opened once a call is made to either PGS_MET_SetFileId or PGS_TD_SetFileId, which are the first functions to be called by the user to establish logical IDs for the file that the user intends to use. The message will be written to the log file every time that PGS_SMF_SetStaticMsg or PGS_SMF_SetDynamicMsg is called in a routine.

Example:

```
(10028) : PGSMET_E_FILETOODL_ERR  
         detected in : PGS_MET_GetPCAttr()  
         unable to convert HDF attribute info on ODL format.
```

A few other SMF tools that are useful in error handling are also explained in this section.

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 6-28. 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 some 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 6-29. 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_Initialization() )
```

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,
  'levellA_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[]);

FORTTRAN: 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 6-30. 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

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 6-31. 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

6.2.3 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 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).

6.2.3.1 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

6.2.3.2 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).

6.2.3.3 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.

6.2.3.4 Toolkit Julian Dates

6.2.3.4.1 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.

6.2.3.4.2 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.

6.2.3.4.3 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:

$$\begin{aligned} &\{2449452.50, 0.0 - 0.0000004972222\} \\ &= \{2449452.50, -0.0000004972222\} \\ &= \{2449451.50, 0.9999995027778\} \end{aligned}$$

6.2.3.5 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.

6.2.3.5.1 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 Greenwich Mean Time to as close an equivalent of UT1 as possible; the Toolkit thus determines Earth rotation from GMT in that period.

6.2.3.5.2 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 ($\sim < 15$ 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. 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 6-32 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 H for an example of a utcpole.dat file.

**Table 6-32. 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, the 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 PCFT, e.g. if a command-line run were possible in processing science data

6.2.3.6 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 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.

6.2.3.7 Time and Date Conversion Tools

Establish Logical IDs for Files to be Used

NAME: PGS_TD_SetFileId()

SYNOPSIS:

C: #include <PGS_TD.h>

PGSt_SMF_status
PGS_TD_SetFileId();

FORTTRAN: include 'PGS_SMF.f'
include 'PGS_tk.f'

integer function pgs_td_setfileid()

DESCRIPTION: This tool sets logical Ids assigned for the user defined files in the PCFT file **filetable.temp**.

INPUTS: None

OUTPUTS: None

RETURNS:

Table 6-33. PGS_TD_SetFileId Returns

Return	Description
PGS_S_SUCCESS	Successful return
PGSTD_E_FILE_OPEN_ERR	Error opening file File_table or LogStatus file

EXAMPLES:

C: PGSt_SMF_status returnStatus;
returnstatus = PGS_TD_SetFileId();
if (returnStatus != PGS_S_SUCCESS)
{
*** do some error handling ***
:
:
}

FORTTRAN: implicit none

```
integer          returnstatus
integer          pgs_td_setfileid
returnstatus = pgs_td_setfileid()
if (returnstatus .ne. pgs_s_success) goto 999
```

NOTES: None

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_tk.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 6-34. 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 6-35. 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 6-36. 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 6.2.3.5.1 (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_tk.f'

          integer function pgs_td_taitoutc(sectai93, asciutc)
            character*27      asciutc
            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 6-37. 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 6-38. 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 6-39. 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 6.2.3.5.1 (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_tk.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 6-40. 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 6-41. 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 6-42. 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      sectTAI93;
      PGSt_double      jdTAI[2];

      jdTAI[0] = 2448989.5;
      jdTAI[1] = 0.0003125;

      sectTAI93 = PGS_TD_TAIjdtotAI(jdTAI);

      /* sectTAI93 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_tk.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 6-43. 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 6-44. PGS_TD_TAItoGAST Outputs

Name	Description	Units	Min	Max
gast	Greenwich Apparent Sidereal Time	radians	0	2Pi

RETURNS:

Table 6-45. 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 occurred

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 6.2.3.4 Time and Date Conversion Tool Notes

TIME BOUNDARIES:

See Section 6.2.3.5.2 (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]);

FORTTRAN:

```
include 'PGS_SMF.f'  
include 'PGS_TD.f'  
include 'PGS_tk.f'
```

integer function pgs_td_utc_to_sctime(spacecrafttag, asciutc, sctime)
 integer spacecrafttag
 character*27 asciutc
 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_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 6-46. 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 Config file. The following line **MUST** be contained in the Config file 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 6.2.3.2 (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 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

OUTPUTS:**Table 6-47. 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 6-48. 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 Config file. The following line **MUST** be contained in the Config file 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 6.2.3.2 (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 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_tk.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 6-49. 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 6-50. 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 6-51. 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 6.2.3.2 (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:cd

```
include 'PGS_SMF.f'
include 'PGS_tk.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 6-52. 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 6-53. 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 6-54. PGS_TD_ASCIItime_BtoA Returns (1 of 2)

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

Table 6-54. PGS_TD_ASCIItime_BtoA Returns (2 of 2)

Return	Description
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 6.2.3.2 (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_tk.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 6-55. 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 6-56. 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 6-57. 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 6.2.3.5.1 (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_tk.f'

integer function pgs_td_gpstoutc(secgps, asciutc)
    double precision    secgps
    character*27        asciutc
```

DESCRIPTION: This tool converts from GPS time to UTC time.

INPUTS:

Table 6-58. 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 6-59. 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 6-60. 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 6.2.3.5.1 (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_tk.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 6-61. 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 6-62. PGS_TD_UTCtoTDTjed Outputs

Name	Description	Units	Min	Max
jedTDT	TDT as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-63. 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 6.2.3.4 (Toolkit Julian Dates)

See Section 6.2.3.2 (ASCII Time Formats)

See See Section 6.2.3.5.1 (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_tk.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 6-64. 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 6-65. PGS_TD_UTCtoTDBjed Outputs

Name	Description	Units	Min	Max
jedTDB	TDB as a Julian date	days	see NOTES	see NOTES

RETURNS:

Table 6-66. 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 6.2.3.2 (ASCII Time Formats)

See Section 6.2.3.4 (Toolkit Julian Dates)

See Section 6.2.3.5.1 (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_tk.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 6-67. 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 6-68. PGS_TD_TimeInterval Outputs

Name	Description	Units	Min	Max
interval	elapsed time interval	seconds	none	none

RETURNS:

Table 6-69. 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_tk.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 6-70. 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 6-71. PGS_TD_UTCtoUTCjd Outputs

Name	Description	Units	Min	Max
jdUTC[2]	UTC Julian date	days	none	none

RETURNS:

Table 6-72. 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 6.2.3.4 (Toolkit Julian Dates)

REQUIREMENTS: PGSTK - 1170, 1220

Convert UTC Julian Date to CCSDS ASCII Time Code A Format

NAME: PGS_TD_UTCjdttoUTC()

SYNOPSIS:

```
C:      #include <PGS_TD.h>

        PGSt_SMF_status
        PGS_TD_UTCjdttoUTC(
            PGSt_double jdUTC[2],
            PGSt_boolean onLeap,
            char         asciiUTC[28])
```

```
FORTRAN: include 'PGS_SMF.f'
          include 'PGS_tk.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 6-73. PGS_TD_UTCjdttoUTC 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 6-74. PGS_TD_UTCjdttoUTC Outputs

Name	Description	Units
asciiUTC	UTC time in CCSDS ASCII Time Code A format	time

RETURNS:

Table 6-75. PGS_TD_UTCjdttoUTC 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(2) = 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 6.2.3.4 (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_tk.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 6-76. 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 6-77. 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 6.2.3.2 (ASCII Time Formats)

See Section 6.2.3.5.2 (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_tk.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 6-78. 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 6-79. 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 6-80. 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 6.2.3.2 (ASCII Time Formats)

See Section 6.2.3.4 (Toolkit Julian Dates)

See Section 6.2.3.5.2 (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_tk.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 6-81. Get Leap Second Inputs

Name	Description	Units	Min	Max
jdUTC	UTC Julian Day number	days (see NOTES)	N/A	N/A

OUTPUTS:

Table 6-82. 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 ACTUAL or 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 6-83. 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 obsolete format was encountered in the Leap Seconds file, if they encounter the “PREDICTED” status.

UTC: Coordinated Universal Time

TAI: International Atomic Time

REQUIREMENTS: PGSTK - 1050, 0930

6.2.3.8 TD Functions

PGS_TD_SetFileId

This tool sets logical file IDs for the files to be used.

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 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_ASCIItime_BtoA

See description in 6.2.3.7 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_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 12AM 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 6.2.3.7 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 6.2.3.7 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 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_TAIjdtoTAI

See description in 6.2.3.7 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 6.2.3.7 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 6.2.3.7 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 6.2.3.7 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 6.2.3.7 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 6.2.3.7 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_UTCjdtotUT1jd

This tool converts UTC time as a Julian date to UT1 time as a Julian date.

PGS_TD_UTCjdtotUTC()

See description in 6.2.3.7 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_UTCtoEOSPMGIIS

This function converts UTC in CCSDS ASCII time code A (or B) format to EOS PM spacecraft (s/c)GIIS 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 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTAI

See description in 6.2.3.7 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 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_UTCtoTDTjed

See description in 6.2.3.7 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 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUT1jd

See description in 6.2.3.7 Time and Date Conversion Tools.

PGS_TD_UTCtoUTCjd

See description in 6.2.3.7 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.

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 Toolkit Tools

A.1.1.2 HDF File Access Tools

- a. The users will be provided the HDF NCSA source codes. They need to install it on their own (for UNIX and Windows 98 and NT) or by install scripts provided with this release (UNIX Platforms only). (HDF distribution is available via anonymous ftp from <ftp.ncsa.uiuc.edu>, 141.142.20.50 or HTTP download from <http://hdf.ncsa.uiuc.edu/rel4links.html>.)

A.1.1.4 Metadata

PGS_MET_Init(), PGS_MET_Init_NonMCF()

- a. A Metadata Configuration File (MCF) or an ASCII file 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 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 PCFT file **filetable.temp** and support 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 rules in creating the PCFT file.

PGS_MET_GetConfig ()

- a. It is assumed that configuration data is held in config file as prescribed by the config file rules of Toolkit (see Appendix F).
- 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 one log file will need to be created by the Toolkit: User Status Log.
- b. Toolkit Errors are directed to the User Status Log file.

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.1.7 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.

This page intentionally left blank.

Appendix B. SMF Usage

B.1 Description

This appendix complements the description on the usage of error reporting SMF tools in Section 6.2.2 . As mentioned before, the toolkit errors are reported in the LogStatus file, which is opened on a call made to PGS_MET_SetFileId or PGS_TD_SetFileId. The error reporting format is similar to the SDP Toolkit error report format. However, the method used to report the error is similar to the method used in HDF. The codes and their explanations are in the file PGS_tk.h in the include directory.

- 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 a return is unwarranted altogether as in a simple mathematical function (e.g., $y = \text{sine}(x)$):

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 Bachus–Nauer 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

```

Appendix C. PCFT Files

The files that are used during runtime must be entered to a Private Customized File Table (PCFT) file. Once it is complete it should be copied to the directory where the user executes his/her code with file name **filetable.temp**. In reality, the PCFT file is a simplified version on the PCF file which is fully explained in the SDP toolkit User's Guide.

This appendix provides a detailed description of how to define PCFT files

C.1 Defining PCFT Files

This section of the appendix discusses the various components of a Private Customized File Table (PCFT). A sample PCFT format is provided, which is used by the test drivers in the test directory. It contains the actual entries required to support the test drivers for both MET and TD tools..

C.1.1 PCFT Components

- **Subject Fields** A PCFT file MUST contain the following subject fields:
 - 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
 - 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
- **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/
Attribute	-	Full UNIX path to Product Attribute file

C.1.2 Format Rules

- All Record fields are placed in the order shown above
- All comments must begin with the pound sign token '#'
- 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 except the last one must be delimited with a pipe token '|'
- A Dummy record with identifier = 0 (as in the sample) must be entered before the last line
- The last line of the file must begin with a subject field token '?'

C.1.3 Sample:

The following file was delivered along with the Toolkit Installation. To access this file, go to directory \$PGSHOME/runtime. This file has all the entries required to run test drivers for MET and TD tools in the directory \$PGSHOME/test (Note: For WINDOWS 95/98/NT one must use “\” instead of “/” for directories)

```
#####
# This file is needed for testing TIME tools. Only the Path for the files
# need to be changed.
#
# The following IDs are defined in the TOOLKIT and they SHOULD NOT be changed
#####
10100|LogStatus|<PGSHOME_PATH>/runtime/LogStatus
5000|configfile.dat|<PGSHOME_PATH>/runtime/configfile.dat
10252|GetAttrtemp|<PGSHOME_PATH>/test/test_MET/GetAttrtemp
10254|MCFWrite.temp|<PGSHOME_PATH>/test/test_MET/MCFWrite.temp
10255|AsciiDump|<PGSHOME_PATH>/test/test_MET/AsciiDump
10256|temporary.MCF|<PGSHOME_PATH>/test/test_MET/temporary.MCF
10301|leapsec.dat|<PGSHOME_PATH>/database/common/TD/leapsec.dat
```

```

10401|utcpole.dat|<PGSHOME_PATH>/database/common/CSC/utcpole.dat
10402|earthfigure.dat|<PGSHOME_PATH>/database/common/CSC/earthfigure.dat
10601|de200.eos|<PGSHOME_PATH>/database/common/CBP/de200.eos
10801|sc_tags.dat|<PGSHOME_PATH>/database/common/EPH/sc_tags.dat
10302|udunits.dat|<PGSHOME_PATH>/database/common/CUC/udunits.dat
#####
# Logical IDs assigned for input/output files can be changed BUT they
# should be different from the IDs assigned above.
#####
10251|data_dict|<PGSHOME_PATH>/test/test_MET/MET_TestData/data_dict
10271|dummy|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFmorahan4
10284|dummy|<PGSHOME_PATH>/test/test_MET/MET_TestData/asciitestfile
5721|hdfestfile|<PGSHOME_PATH>/test/test_MET/MET_TestData/hdfestfile
5722|hdfestfile_5722|<PGSHOME_PATH>/test/test_MET/MET_TestData/hdfestfile_5722
5724|hdfestfile_5724|<PGSHOME_PATH>/test/test_MET/MET_TestData/hdfestfile_5724
5725|hdfestfile_5725|<PGSHOME_PATH>/test/test_MET/MET_TestData/hdfestfile_5725
5728|LISUSR|<PGSHOME_PATH>/test/test_MET/MET_TestData/LISUSR
5729|LIS_FILTERED|<PGSHOME_PATH>/test/test_MET/MET_TestData/LIS_FILTERED
5730|hdfestfile_5730|<PGSHOME_PATH>/test/test_MET/MET_TestData/hdfestfile_5730
5731|ascii_input|<PGSHOME_PATH>/test/test_MET/MET_TestData/MOP_THRESH
5030|MCFfile|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFfile
5031|MCFfile_1|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFfile_1
5033|MCFfile_3|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFfile_3
5036|MCFfile_6|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFfile_6
5038|MCFfile_8|<PGSHOME_PATH>/test/test_MET/MET_TestData/MCFfile_8
#####
# files to check PGS_MET_InitNonMCF function
#####
5800|MOD10_L2_ASCII.met|<PGSHOME_PATH>/test/test_MET/MET_TestData/MOD10_L
2_ASCII.met

```

```
5801|morahan44_ascii/<PGSHOME_PATH>/test/test_MET/MET_TestData/morahan44_ascii
5802|MOD10_L2_HDF1.hdf/<PGSHOME_PATH>/test/test_MET/MOD10_L2_HDF1.hdf
5803|MOD10_L2_HDF2.hdf/<PGSHOME_PATH>/test/test_MET/MOD10_L2_HDF2.hdf
5804|NAT_File/<PGSHOME_PATH>/test/test_MET/NAT_File
#####
# End this table with next two lines. Last line should be ?
#####
0|DUMMY/<PGSHOME_PATH>/test/test_MET/MET_TestData/DUMMY
?
```

Appendix D. Population of Granule Level Metadata Using the Metadata Tools

D.1 Introduction

The purpose of this appendix is to provide detailed guidance on the use of the 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 D.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 D.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 D.4 describes how metadata is generated and written to output files using the toolkit. Section D.4 also includes a discussion of how HDF and non-HDF product files are treated differently. Section D.5 discusses metadata toolkit usage. Section D.6 describes in detail the structure and syntax of the MCF. Section D.7 discusses metadata in HDF vs. non-HDF input and Output Files.

D.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-002-005 ("*Science Data Processing Segment Database Design and Database Schema Specification for the ECS Project, 5/96*"); and updates for the ECS Release B.0.: 420-TP-015-002 ("*B.0 Implementation Earth Science Data Model for the ECS Project*") and 420-TP-016-001 ("*Backus-Naur Format (BNF) Representation of the B.0 Earth Science Data Model for the ECS Project*".)

D.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 D.6.4.

D.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 D-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 D.3 for roles and responsibilities for preparation of the collection and granule metadata.

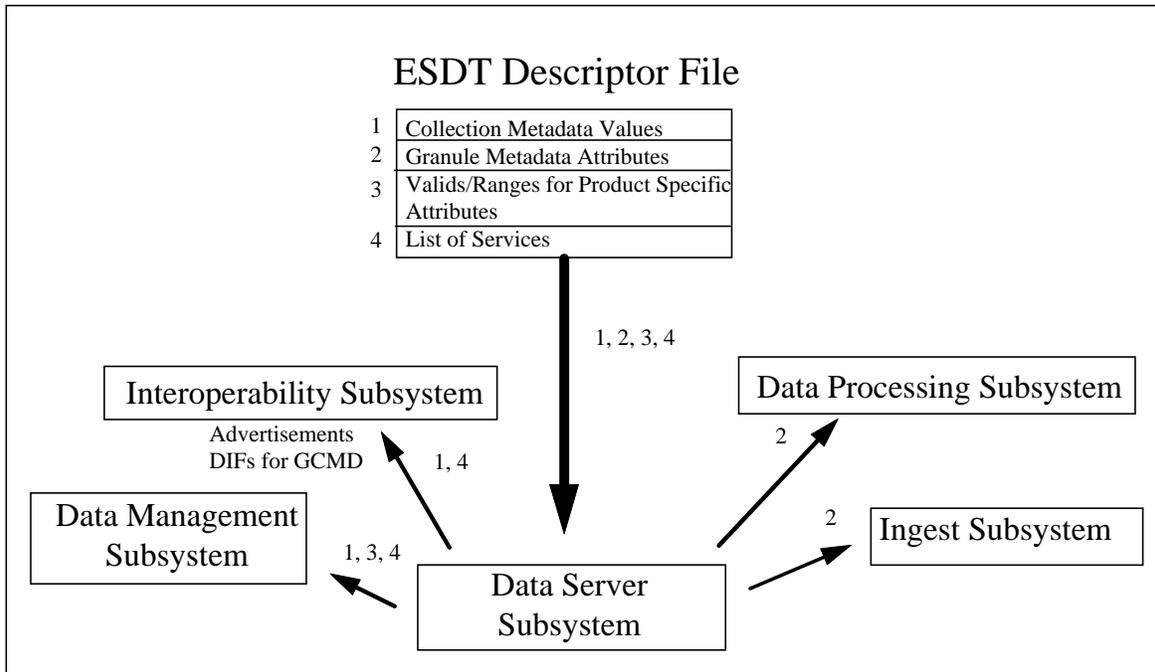


Figure D-1. Metadata Flow in ECS

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.

D.2.3 Mandatory Metadata

In 420-TP-016-001 (“*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 D.6.2.

D.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 this 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. Details are provided for the process of generating an MCF file in section D.3.1. 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 metadata@eos.hitc.com. Specific questions regarding metadata or ESDTs may also be sent to this email address.

D.3.1 Generating the Metadata Configuration File in Release B.0

Beginning with B.0, ECS provides you with a Metadata Configuration File built from the tool MetaData Works . Not only does this free your team from mastering the Object Description Language (ODL), but it also provides you with an ECS-validated Metadata Configuration File.

The following are the steps in utilizing MetaData Works to build a new MCF file:

1. Using a JAVA capable web browser (e.g., Netscape 3.0), go to URL <http://et3ws1.hitc.com/>
2. Read the information on the welcome page, and then select MetaData Works.
3. As shown in Figure D-2, click on the “Select Type” button.
4. On the next Web Page, entitled “Data Type: ESDTDescriptor”, enter the ESDT Descriptor Name, select the target level of metadata (Full, Intermediate or Limited) coverage from the pull-down menu, and click on the button “Create New File”
5. On the next Web Page, entitled “New ESDTDescriptor Defaults Selection”, select the “No Defaults” radio button and then click on the “Create/Edit This” button.

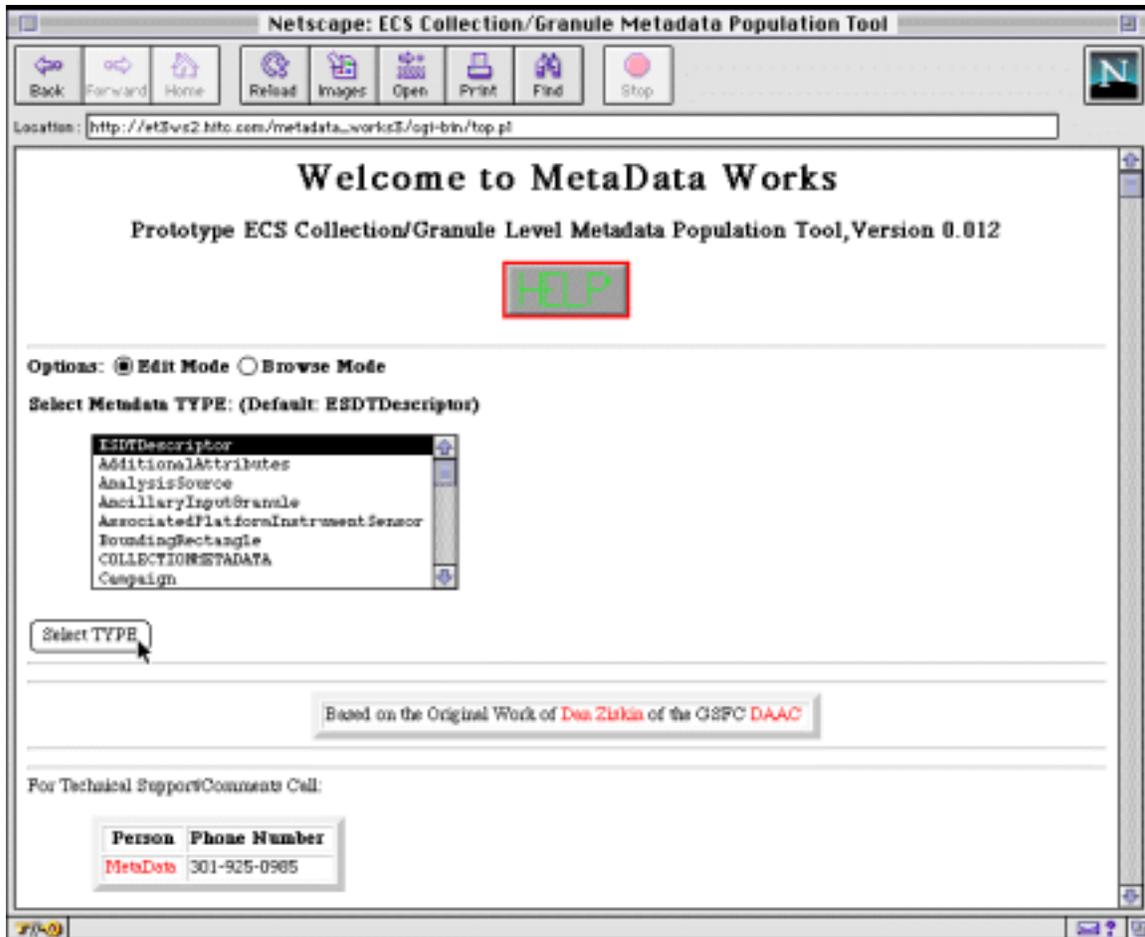


Figure D-2. Metadata Works Welcome Page

6. The next Web Page, shown in Figure D-3, is the form for selecting the granule-level attributes. Clicking on any of the small blue filled circles to the left of the attributes brings up a definition and other information. Those attributes that are mandatory at the level of metadata coverage selected in step 4 are automatically included in the MCF. The optional attributes are shown with check boxes next to them. The attributes which can be multiple also have a text entry field where the maximum number of values that will be set by the PGE needs to be indicated.
7. Enter the ESDT ShortName and VersionID in the text entry fields.
8. Click in the Check Boxes for those optional attributes which are desired.
9. Enter the maximum number of values to be set by the PGE for selected attributes for which multiple values may be set.

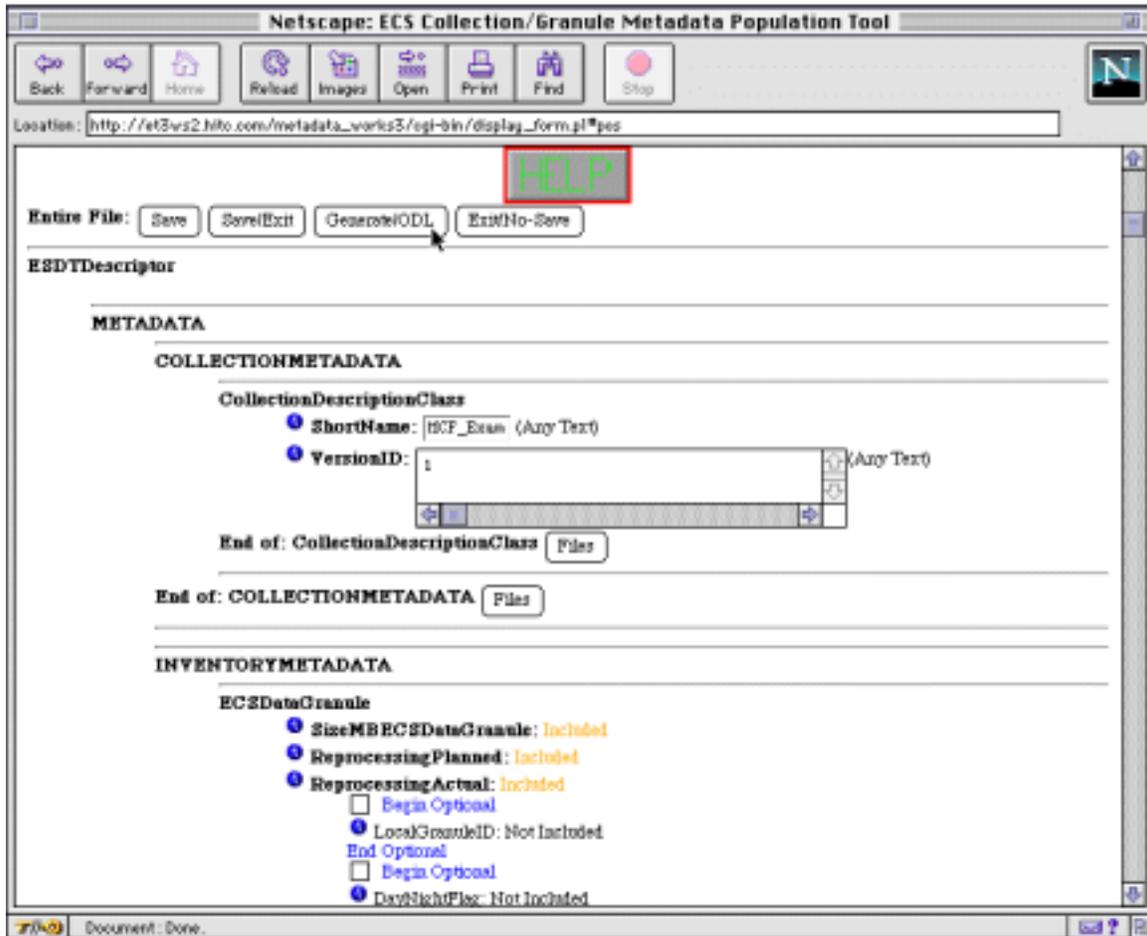


Figure D-3. MCF Attribute Selection Web Page

10. For temporal and/or spatial attribute, select the radio button for the desired group.
11. When all entry is complete, Click on the “Save” button.
12. To generate an MCF, first click on the “Generate ODL” button.
13. On the next Web Page showing the ODL Generation Progress, click on the “Display MCF File” button. The ODL comprising the MCF file is displayed, as shown in Figure D-4.
14. To use the generated ODL as an MCF in your testing, save this Web Page as Text to your local storage.

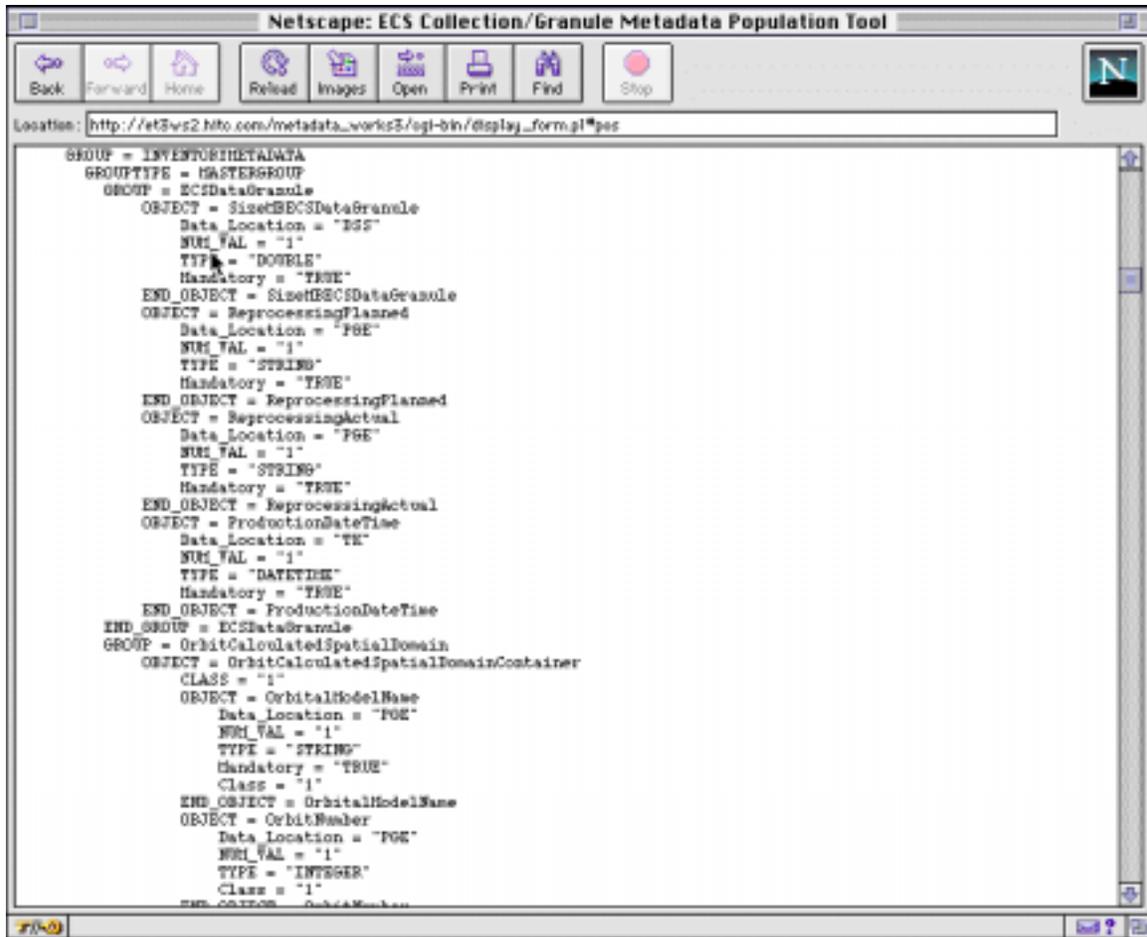


Figure D-4. Example of Generated MCF Object Description Language

D.4 The Granule Metadata Population Process

Figure D-5 is a schematic of the process by which data granules and their metadata are generated. In Step 1 Science Data Server notifies Science 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 PCFT 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 D.5 and D.6 of this Appendix.

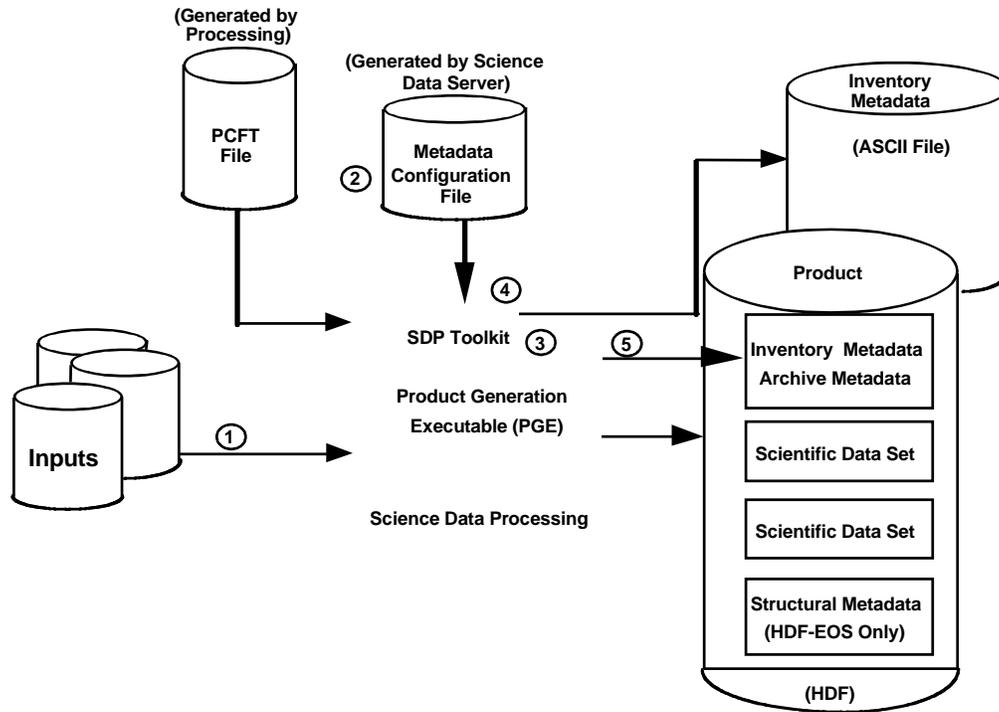


Figure D-5. Science Data Production and Archival Scenario.

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.

D.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.

D.5.1 Overview

Multiple MCFs may be opened and written to from within a single PGE. It is also possible to open metadata files where the attribute values are written as ASCII records. There are four metadata tools that are used in conjunction with an MCF or ASCII metadata file that must be called in a specific sequence for each MCF. First, each MCF or ASCII must be initialized with **PGS_MET_Init** or **PGS_MET_Init_NonMCF**. Each call to **PGS_MET_Init** or **PGS_MET_Init_NonMCF** returns a unique identifier for that MCF or the temporary MCF that is created from the ASCII metadata file. 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. Before writing metadata to an HDF file PGE must open HDF file for writing. If the HDF file is of HDF4 type this can be accomplished by calling HDF's function **SDstart** (or **sfstart** for FORTRAN). If the HDF file is of type HDF5 the PGE must call **PGS_MET_SDstart** (this can also be used for HDF files of type HDF4). Finally, **PGS_MET_Remove** frees up memory occupied by the MCFs and **PGS_MET_SDend** closes HDF file.

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 config file discussed in Appendix F.

D.5.2 Example 1

This example includes retrieval of metadata from an HDF file and from the config file, 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 D.6.

First the function **PGS_MET_SetFileId()** is called to open and read the contents of the file **filetable.temp** that includes entries for the files to be used (See Appendix C for sample)

```
/* Set File Ids for input/output files */
ret = PGS_MET_SetFileId();
```

Next a value for the runtime parameter with the name "Runtime_ID" is read from the user-defined runtime parameters section of the Config file using **PGS_MET_GetConfigData**:

```
/* get values from Config file */
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 concerning specification of metadata input files in the PCFT.)

```
/* 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 and D.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 D.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 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.

```

/* open the HDF file of type HDF4 for writing metadata */
sdid1 = SDstart("HDF4_File.hdf", DFACC_RDWR);

/* Write Metadata Blocks to HDF4 output file */
ret =
PGS_MET_Write(mdHandles[1], "coremetadata", sdid1);

ret =
PGS_MET_Write(mdHandles[2], "archivemetadata", sdid1);

/* Write all Metadata Blocks to ASCII output file */
ret =
PGS_MET_Write(mdHandles[0], NULL, 101);

/* open the HDF file of type HDF5 for writing metadata */
ret =
PGS_MET_SDstart("HDF5_File.h5", H5F_ACC_RDWR, &sdid2);

/* Write Metadata Blocks to HDF5 output file */
ret =
PGS_MET_Write(mdHandles[1], "coremetadata", sdid2);

ret =
PGS_MET_Write(mdHandles[2], "archivemetadata", sdid2);

/* Remove MCF from memory and close HDF files */
ret =
PGS_MET_Remove();

(void) SDend(sdid1);

(void) PGS_MET_SDend(sdid2);

```

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.

D.5.3 Example 2

Users can create their own metadata file with attribute values entered as ASCII records. This example includes initializing such an ASCII metadata file, reading its contents into the memory after creating a temporary MCF file and then writing it to an HDF file or another ASCII file. A sample ASCII file and the MCF file generated from it is presented in Section D9. As in Example 1, the code fragments are in C.

First the function `PGS_MET_SetFileId` is called to set file IDs for input/output files

```
/* Set file Ids for input/output files */
ret = PGS_MET_SetFileId();
```

Next, `PGS_MET_Init_NonMCF` is used to read the contents of the ASCII metadata file `morahan44_ascii` whose file ID is 5801. The returned `mdHandles` will point to the temporary MCF file created with this call

```
/* Initialize an ASCII metadata file */
ret = PGS_MET_Init_NonMCF(5801, mdHandles);
```

Except the object “ProductionDateTime” which will be set by the Toolkit, values for other objects are set in the ASCII file and cannot be changed by a call to `PGS_MET_SetAttr`.

Once the metadata is read to the memory, the inventory metadata can be written to the product. It can also be written to another ASCII output.

```
/* open the HDF file of type HDF4 for writing metadata */
sdid1 = SDstart("HDF4_File.hdf", DFACC_RDWR);

/*Write metadata to HDF output file with id sdid1*/
ret = PGS_MET_Write(mdHandles[1], "coremetadata", sdid1);

/* Wrote metadata to ASCII output file with id 5804 */
ret = PGS_MET_Write(mdHandles[0], NULL, 5804);

/* open the HDF file of type HDF5 for writing metadata */
ret =
PGS_MET_SDstart("HDF5_File.h5", H5F_ACC_RDWR, &sdid2);

/* Write Metadata Blocks to HDF5 output file */
ret =
PGS_MET_Write(mdHandles[1], "coremetadata", sdid2);

/* Remove MCF from memory and close HDF files */
ret =
PGS_MET_Remove();

(void) SDend(sdid1);

(void) PGS_MET_SDend(sdid2);
```

D.6 Structure of the Metadata Configuration File (MCF)

As described in Section D.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.

D.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 = ARCHIVEMETADATA
      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.

D.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 in the Config file 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 Config file, and its corresponding value between the second and third delimiters . (For further details on the format of the Config file, see Appendix C of this document.) For example, if the Config file contained:

```
10255 | PLATFORMSHORTNAME | "TRMM"
```

then

```
ret = PGS_MET_GetConfigData("PLATFORMSHORTNAME", &val)
```

would return “TRMM” in val. In the Config file quotes are only necessary when the datatype of the value in the MCF is STRING. If an attribute is to be stored in the Config file as a runtime parameter, the attribute name must be in UPPER case and must appear only once in the Config file.

“**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.

D.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-016-001). 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 SensorCharacteristicValue 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 SensorCharacteristicDataType would anyone someone to convert the string returned for SensorCharacteristicValue 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.

D.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 AdditionalAttribute, 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.

D.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 in the manner described in main body of the Toolkit documentation.

D.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.

Additional information on this notation can be found at WWW address http://bolero.gsfc.nasa.gov/ccsds/ccsds_document_access.html.

- 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

D.9 ASCII Metadata file and resultant MCF

Following is a sample ASCII metadata file that can be read by PGS_MET_Init_NonMCF. This function in turn creates an MCF file and initializes it as a normal MCF file. The sample MCF generated from the sample ASCII file is presented after the ASCII file. Note that in order to be able to write the metadata to product file using PGS_MET_Write, we should have the object ProductionDateTime as shown in the sample. The value entered for this object will be set by the Toolkit at the time of writing the metadata into a product file. Without this object PGS_MET_Write will return an error.

ASCII Metadata File

```
GROUP=LEVEL0METADATA
OBJECT = ProductionDateTime
  VALUE = 1998-04-06T19:24:37.000
  NUM_VAL = 1
  TYPE = "DATETIME"
END_OBJECT = ProductionDateTime
OBJECT=VOYAGER_02
  Value= "g3aexpm1"
  Num_Val=1
  Type= "STRING"
END_OBJECT=VOYAGER_02
OBJECT=Data_Start_Time
  Value= 1997-01-29T15:31:02.1234
  Num_Val=1
  Type= DATETIME
END_OBJECT=Data_Start_Time
OBJECT=Data_End_Time
  Value= 1997-07-28T12:00:00.123456
  Num_Val=1
  Type= DATETIME
END_OBJECT=Data_End_Time
OBJECT=SizeMBECSDataGranule
  Value=20
  Num_Val=1
  Type="INTEGER"
END_OBJECT=SizeMBECSDataGranule
OBJECT=Overflow_flag
  Value= 31459
```

```

        Num_Val=1
        Type="INTEGER"
END_OBJECT=Overflow_flag
OBJECT=Total_NE
    Value= 3.1459e2
    Num_Val=1
    Type="DOUBLE"
END_OBJECT=Total_NE
OBJECT=Total_SS
    Value= 222.33E-2
    Num_Val=1
    Type="DOUBLE"
END_OBJECT=Total_SS
OBJECT=Total_SR
    Value= 1
    Num_Val=1
    Type="INTEGER"
END_OBJECT=Total_SR
OBJECT=Total_MR
    Value= 0
    Num_Val=1
    Type="INTEGER"
END_OBJECT=Total_MR
OBJECT=Total_MS
    Value= 0
    Num_Val=1
    Type="INTEGER"
END_OBJECT=Total_MS
OBJECT=Special_Calibration
    Value= 0
    Num_Val=1
    Type="INTEGER"
END_OBJECT=Special_Calibration
OBJECT=Research_mode
    Value= 0
    Num_Val=1
    Type="INTEGER"
END_OBJECT=Research_mode
OBJECT=Begin_Orbit_Number
    Value= "SHOT_1_RANGE_TO_SUEFACE"
    Num_Val=1
    Type="STRING"
END_OBJECT=Begin_Orbit_Number
OBJECT=End_Orbit_Number
    Value= "(null) 4206770 %3B"
    Num_Val=1
    Type="STRING"
END_OBJECT=End_Orbit_Number
OBJECT=USA_NASA
    Value= "g3aexpm2"
    Num_Val=1
    Type= "STRING"
END_OBJECT=USA_NASA
OBJECT=SHOT_1
    Value= "g3aexpm3"
    Num_Val=1
    Type= "STRING"
END_OBJECT=SHOT_1
OBJECT=SHOT_2
    Value= "g3aexpm3"
    Num_Val=1
    Type= "STRING"

```

END_OBJECT=SHOT_2
END_GROUP=LEVEL0METADATA
END

Temporary MCF File Created from ASCII Metadata File

```
GROUP = INVENTORYMETADATA
GROUPTYPE = MASTERGROUP

OBJECT = PRODUCTIONDATETIME
VALUE = 1998-04-06T19:24:37.000Z
NUM_VAL = 1
TYPE = "DATETIME"
DATA_LOCATION = TK
MANDATORY = FALSE
END_OBJECT = PRODUCTIONDATETIME

OBJECT = VOYAGER_02
VALUE = "g3aexpm1"
NUM_VAL = 1
TYPE = "STRING"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = VOYAGER_02

OBJECT = DATA_START_TIME
VALUE = 1997-01-29T15:31:02.1234Z
NUM_VAL = 1
TYPE = DATETIME
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = DATA_START_TIME

OBJECT = DATA_END_TIME
VALUE = 1997-07-28T12:00:00.123456Z
NUM_VAL = 1
TYPE = DATETIME
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = DATA_END_TIME

OBJECT = SIZEMBECSDATAGRANULE
VALUE = 20
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = SIZEMBECSDATAGRANULE

OBJECT = OVERFLOW_FLAG
VALUE = 31459
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = OVERFLOW_FLAG

OBJECT = TOTAL_NE
```

```

VALUE = 3.1459e+02
NUM_VAL = 1
TYPE = "DOUBLE"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = TOTAL_NE

OBJECT = TOTAL_SS
VALUE = 2.22e+00
NUM_VAL = 1
TYPE = "DOUBLE"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = TOTAL_SS

OBJECT = TOTAL_SR
VALUE = 1
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = TOTAL_SR

OBJECT = TOTAL_MR
VALUE = 0
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = TOTAL_MR

OBJECT = TOTAL_MS
VALUE = 0
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = TOTAL_MS

OBJECT = SPECIAL_CALIBRATION
VALUE = 0
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = SPECIAL_CALIBRATION

OBJECT = RESEARCH_MODE
VALUE = 0
NUM_VAL = 1
TYPE = "INTEGER"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = RESEARCH_MODE

OBJECT = BEGIN_ORBIT_NUMBER
VALUE = "SHOT_1_RANGE_TO_SUEFACE"
NUM_VAL = 1
TYPE = "STRING"
DATA_LOCATION = MCF
MANDATORY = FALSE
END_OBJECT = BEGIN_ORBIT_NUMBER

```

```

OBJECT          = END_ORBIT_NUMBER
  VALUE         = "(null) 4206770 %3B"
  NUM_VAL       = 1
  TYPE          = "STRING"
  DATA_LOCATION = MCF
  MANDATORY     = FALSE
END_OBJECT      = END_ORBIT_NUMBER

OBJECT          = USA_NASA
  VALUE         = "g3aexpm2"
  NUM_VAL       = 1
  TYPE          = "STRING"
  DATA_LOCATION = MCF
  MANDATORY     = FALSE
END_OBJECT      = USA_NASA

OBJECT          = SHOT_1
  VALUE         = "g3aexpm3"
  NUM_VAL       = 1
  TYPE          = "STRING"
  DATA_LOCATION = MCF
  MANDATORY     = FALSE
END_OBJECT      = SHOT_1

OBJECT          = SHOT_2
  VALUE         = "g3aexpm3"
  NUM_VAL       = 1
  TYPE          = "STRING"
  DATA_LOCATION = MCF
  MANDATORY     = FALSE
END_OBJECT      = SHOT_2

END_GROUP      = INVENTORYMETADATA

END

```

This page intentionally left blank.

Appendix E. Test Drivers

The Toolkit_MTD is delivered with a series of test drivers for metadata and time/date tools. These test programs are provided to aid users in the development of software using the Toolkit_MTD. The user may run the same test cases as included in the directories \$PGSHOME/test/test_MET or \$PGSHOME/test/test_TIME, to verify that the Toolkit_MTD is functioning correctly. The directories contain source codes for a driver in C and Fortran for all tools, metadata and time/date, makefiles for the tool groups, README files explaining how to use the drivers, **filetable.temp** showing the files needed to run the drivers, sample input files and sample output files. The input/output files required by these drivers are tabulated in the **filetable.temp**, that needs to be edited to correct the path for the required files (see Appendix C.) To compile and run the test drivers, follow the steps mentioned in the README files in the test subdirectories test_MET and/or test_TIME. The output files from the drivers can be checked against the sample outputs using UNIX “diff” command to assure that the tools are installed and working properly.

In a UNIX environment one can also use the script runTest (or runTest.cpp for C++ installed Toolkit MTD) in the directory \$PGSHOME/test/Common. The script runs all test programs in test_MET and test_TIME directories, producing *.diff files between outputs and sample outputs. If the script is used for running test programs, there is no need for editing filetable.temp prior to executing runTest (or runTest.cpp), where the script will produce correct filetable.temp for the test programs. Please see the file README.script in the directory \$PGSHOME/test/Common for details.

The following is the listing for a C program that demonstrates the use of functions for MET tools. The PCFT file used for this example is followed by this listing.

```

/*****
BEGIN_FILE_PROLOG:

FILENAME: PGS_MET_example.c

DESCRIPTION:
    This file contains the test driver for the following functions :

        PGS_MET_Init()
        PGS_MET_Init_NonMCF()
        PGS_MET_SetFileId()
        PGS_MET_GetFileId()
        PGS_MET_GetConfigData()
        PGS_MET_GetPCAttr()
        PGS_MET_SetAttr()
        PGS_MET_GetSetAttr()
        PGS_MET_Write()
        PGS_MET_Remove()

    it also calls the following functions:

        SDstart()
        SDend()

INPUTS:
    Input files used in this example are in the directory

```

.../TOOLKIT_MTD/test/test_MET/MET_TestData
 The PCFT file used for this example is in the same directory
 (PCFT_TOOLKIT_MTD_MET_example) and followed this code listing.

AUTHORS :
 Abe Taaheri / Space Applications Corporation

HISTORY :
 1-Jun-1998 AT Original version

```

END_FILE_PROLOG:
*****/
/* include files */

#include <PGS_MET.h>
#include <PGS_tk.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hdf.h>
#include <mfhdf.h>
#include <PGS_SMF.h>

#define INVENTORYMETADATA 1
#define ARCHIVEMETADATA 2
#define ODL_IN_MEMORY 0

extern PGSt_SMF_status
PGS_PC_GetReference(PGSt_MET_Logical prodID, PGSt_integer *version,char
                  *referenceID);

int main()
{
/*****
Declarations.
*****/

PGSt_MET_all_handles    mdHandles;
PGSt_MET_all_handles handles;
char fileName1[PGSd_MET_FILE_PATH_MAX]="";
char fileName2[PGSd_MET_FILE_PATH_MAX]="";
char my_HDF_file[PGSd_MET_FILE_PATH_MAX]="";
char my_HDF5_file[PGSd_MET_FILE_PATH_MAX]="";
char          msg[PGS_SMF_MAX_MSG_SIZE];
char          mnemonic[PGS_SMF_MAX_MNEMONIC_SIZE];
char          fileMessage[PGS_SMF_MAX_MSG_SIZE];
char
myFile[PGSd_MET_FILE_PATH_MAX]="/net/cherokee/DEM/stx_MET_TIME/test/SCF_METADATA_TOOLS
/test/test_MET/MET_TestData/MODIS_FILE.hdf";
int32 sdid1;
PGSt_integer sdid5;
PGSt_SMF_status ret = PGS_S_SUCCESS;
char *informationname;
PGSt_integer ival =3;
PGSt_double dval=203.2;

PGSt_integer fileId, fileId2;

PGSt_integer i;
PGSt_integer          version;
PGSt_SMF_status      returnStatus;
char *mysaval[5];

/*****
/* Associate logical IDs with physical filenames. */
*****/

```

```

ret=PGS_MET_SetFileId();
printf("ret after PGS_MET_SetFileId()is %d in Main\n",ret);

if(ret != PGS_S_SUCCESS)
{
    printf(" Failed in assigning logical IDs\n");
}

/*recover file name for fileId=PGSd_MET_MCF_FILE */

version = 1;
fileId = PGSd_MET_MCF_FILE;
returnStatus = PGS_PC_GetReference( fileId,
                                   &version,
                                   fileName1);

if ( returnStatus != PGS_S_SUCCESS )
{

    PGS_SMF_GetMsg( &returnStatus, mnemonic, msg );

    if (returnStatus != PGS_S_SUCCESS)
    {
        strcpy(fileMessage, msg);
        PGS_SMF_SetDynamicMsg( returnStatus,
                               fileMessage,
                               "metatest" );
    }
}
else
{
    printf("The input file for ID %d is %s\n",fileId,fileName1);
}

/* test PGS_MET_GetFileId for recovering file ID */

fileId2 = PGS_MET_GetFileId(myFile);
if(fileId != 0)
{
    version = 1;
    returnStatus = PGS_PC_GetReference( fileId2,
                                       &version,
                                       fileName2);

    if ( returnStatus != PGS_S_SUCCESS )
    {
        PGS_SMF_GetMsg( &returnStatus, mnemonic, msg );
        if (returnStatus != PGS_S_SUCCESS)
        {
            strcpy(fileMessage, msg);

            PGS_SMF_SetDynamicMsg( returnStatus,
                                   fileMessage,
                                   "metatest" );
        }
    }
    else
    {
        printf("The input file for ID %d is %s\n",fileId2,fileName2);
    }
}

informationname=(char *) malloc(330);

/* Test PGS_MET_GetPCAttr */

for (i=0;i<5;i++)
{
    mysaval[i]=(char *) malloc(330);
    strcpy(mysaval[i], "");
}

```

```

}

fileId = 5039;
ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
                      "ReprocessingPlanned",mysaval);

if(ret !=PGS_S_SUCCESS)
{
    printf("ReprocessingPlanned after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("ReprocessingPlanned after PGS_MET_GetPCAttr is successful:
%s\n",mysaval);
    for(i = 0; i<5; i++) printf("%s ", mysaval[i]);
}
printf("\n");

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
                      "REPROCESSINGACTUAL",&informationname);

if(ret !=PGS_S_SUCCESS)
{
    printf("REPROCESSINGACTUAL after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("REPROCESSINGACTUAL after PGS_MET_GetPCAttr is successful:
%s\n",informationname);
}

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
                      "QAPercentMissingData.1",&ival);

if(ret !=PGS_S_SUCCESS)
{
    printf("QAPERCENTMISSINGDATA after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("QAPERCENTMISSINGDATA after PGS_MET_GetPCAttr is successful: %d\n",ival);
}

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
                      "WESTBOUNDINGCOORDINATE",&dval);

if(ret !=PGS_S_SUCCESS)
{
    printf("WESTBOUNDINGCOORDINATE after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("WESTBOUNDINGCOORDINATE after PGS_MET_GetPCAttr is successful:
%lf\n",dval);
}

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
                      "EastBoundingCoordinate",&dval);

if(ret !=PGS_S_SUCCESS)
{
    printf("EastBoundingCoordinate after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("EastBoundingCoordinate after PGS_MET_GetPCAttr is successful:
%lf\n",dval);
}

```

```

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
    "SouthBoundingCoordinate",&dval);

if(ret !=PGS_S_SUCCESS)
{
    printf("SouthBoundingCoordinate after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("SouthBoundingCoordinate after PGS_MET_GetPCAttr is successful:
%lf\n",dval);
}

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
    "NorthBoundingCoordinate",&dval);

if(ret !=PGS_S_SUCCESS)
{
    printf("NorthBoundingCoordinate after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("NorthBoundingCoordinate after PGS_MET_GetPCAttr is successful:
%lf\n",dval);
}

ret=PGS_MET_GetPCAttr(fileId,1,"coremetadata.0",
    "RangeBeginningDateTime",&informationname);

if(ret !=PGS_S_SUCCESS)
{
    printf("RangeBeginningDateTime after PGS_MET_GetPCAttr is failed\n");
}
else
{
    printf("RangeBeginningDateTime after PGS_MET_GetPCAttr is successful:
%s\n",informationname);
}

/* test PGS_MET_GetPCAttr with archivemetadata */

ret = PGS_MET_GetPCAttr(fileId2, 1, "archivemetadata", "WestBoundingCoordinate",
&dval);
if(ret == PGS_S_SUCCESS)
{
    printf("dval for WestBoundingCoordinate in ARCHIVEDMETADATA is:%lf ", dval);
    printf("\n");
}
else
{
    printf("EastBoundingCoordinate in ARCHIVEDMETADATA after PGS_MET_GetPCAttr is
failed\n");
}

ret = PGS_MET_GetPCAttr(fileId2, 1, "archivemetadata", "EastBoundingCoordinate",
&dval);
if(ret == PGS_S_SUCCESS)
{
    printf("dval for EastBoundingCoordinate in ARCHIVEDMETADATA is:%lf ", dval);
    printf("\n");
}
else
{
    printf("EastBoundingCoordinate in ARCHIVEDMETADATA after PGS_MET_GetPCAttr is
failed\n");
}

```

```

ret = PGS_MET_GetPCAttr(fileId2, 1, "archivemetadata", "SouthBoundingCoordinate",
&dval);
if(ret == PGS_S_SUCCESS)
{
    printf("dval for SouthBoundingCoordinate in ARCHIVEDMETADATA is:%lf ", dval);
    printf("\n");
}
else
{
    printf("SouthBoundingCoordinate in ARCHIVEDMETADATA after PGS_MET_GetPCAttr is
failed\n");
}

ret = PGS_MET_GetPCAttr(fileId2, 1, "archivemetadata", "NorthBoundingCoordinate",
&dval);
if(ret == PGS_S_SUCCESS)
{
    printf("dval for NorthBoundingCoordinate in ARCHIVEDMETADATA is:%lf ", dval);
    printf("\n");
}
else
{
    printf("NorthBoundingCoordinate in ARCHIVEDMETADATA after PGS_MET_GetPCAttr is
failed\n");
}

/* Initialize MCF file */

fileId = 10250;
ret=PGS_MET_Init(fileId,handles);

if (ret !=PGS_S_SUCCESS)
{
    printf("initialization failed\n");
    return 0;
}
else
{
    printf("ret after PGS_MET_Init is %d\n",ret);
}

/* test PGS_MET_SetAttr */

ival=667788;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"QAPERCENTINTERPOLATEDDATA.1",&ival);
printf("ret after SetAttr for QAPERCENTINTERPOLATEDDATA.1 is %d\n",ret);

ival=12345;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"QAPercentMissingData.1",&ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n",ret);

ival=123;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"QAPercentOutOfBoundsData.1",&ival);
printf("ret after SetAttr for QAPercentOutOfBoundsData.1 is %d\n",ret);

ival=23456;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"QAPercentOutOfBoundsData.2",&ival);
printf("ret after SetAttr for QAPercentOutOfBoundsData.1 is %d\n",ret);

ival=56789;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
"QAPercentMissingData.2",&ival);
printf("ret after SetAttr for QAPercentMissingData.1 is %d\n",ret);

strcpy(informationname,"Exercisel");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],

```

```

        "AutomaticQualityFlagExplanation.1",&informationname);
printf("ret after SetAttr for AutomaticQualityFlagExplanation.1 is %d\n",ret);

strcpy(informationname,"1997/12/23");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "RangeBeginningDateTime",&informationname);
printf("ret after SetAttr for RangeBeginningDateTime is %d\n",ret);

strcpy(informationname,"1997.07/30");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "RangeBeginningDate",&informationname);
printf("ret after SetAttr for RangeBeginningDate is %d\n",ret);

strcpy(informationname,"ReprocessingplannINVENTReprocessingplannINVENTReprocessingplan
nINVENT");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingPlanned",&informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n",ret);

strcpy(informationname,"\"ReprocessingplannARCHIVE");
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "ReprocessingPlanned",&informationname);
printf("ret after SetAttr for ReprocessingPlanned is %d\n",ret);

strcpy(informationname,"Reprocessin");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ReprocessingActual",&informationname);
printf("ret after SetAttr for ReprocessingActual is %d\n",ret);

strcpy(informationname,"ID1111");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "LocalGranuleID",&informationname);
printf("ret after SetAttr for LocalGranuleID is %d\n",ret);

strcpy(informationname,"version1234");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "LocalVersionID",&informationname);
printf("ret after SetAttr for LocalVersionID is %d\n",ret);

strcpy(informationname,"Flag1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag",&informationname);
printf("ret after SetAttr for DayNightFlag is %d\n",ret);

strcpy(informationname,"Flag1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "DayNightFlag",&informationname);
printf("ret after SetAttr for DayNightFlag is %d\n",ret);

strcpy(informationname,"information1");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.1",&informationname);
printf("ret after SetAttr for ParameterName is %d\n",ret);

strcpy(informationname,"information2");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.2",&informationname);
printf("ret after SetAttr for ParameterName.2 is %d\n",ret);

strcpy(informationname,"information3");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "ParameterName.3",&informationname);
printf("ret after SetAttr for ParameterName is %d\n",ret);

strcpy(informationname,"information4");
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],

```

```

        "ParameterName.4",&informationname);
printf("ret after SetAttr for ParameterName is %d\n",ret);

dval=111.11;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "WestBoundingCoordinate",&dval);
printf("ret WestBoundingCoordinate is %d %f\n",ret,dval);

dval=222.22;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "northBoundingCoordinate",&dval);
printf("ret northBoundingCoordinate is %d %f\n",ret,dval);

dval=333.33;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "EastBoundingCoordinate",&dval);
printf("ret EastBoundingCoordinate is %d %f\n",ret,dval);

dval=444.44;
ret=PGS_MET_SetAttr(handles[ARCHIVEDMETADATA],
    "SouthBoundingCoordinate",&dval);
printf("ret SouthBoundingCoordinate is %d %f\n",ret,dval);

dval=11.11;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "WestBoundingCoordinate",&dval);
printf("ret WestBoundingCoordinate is %d %f\n",ret,dval);

dval=22.22;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "northBoundingCoordinate",&dval);
printf("ret northBoundingCoordinate is %d %f\n",ret,dval);

dval=33.33;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "EastBoundingCoordinate",&dval);
printf("ret EastBoundingCoordinate is %d %f\n",ret,dval);

dval=44.44;
ret=PGS_MET_SetAttr(handles[INVENTORYMETADATA],
    "SouthBoundingCoordinate",&dval);
printf("ret SouthBoundingCoordinate is %d %f\n",ret,dval);

/* Get the value of set attribute */

dval=11.11;
ret=PGS_MET_GetSetAttr(handles[INVENTORYMETADATA],
    "SouthBoundingCoordinate",&dval);

printf("after    PGS_MET_GetSetAttr:    ret    SouthBoundingCoordinate    is    %d
%f\n",ret,dval);

/* Get data from config file */

ret = PGS_MET_GetConfigData("TEST_PARM_FLOAT", &dval);
printf("after PGS_MET_GetConfigData : ret TEST_PARM_INT is %d %f\n",ret, dval);

/* write metadata to HDF4 and ASCII files */
version =1;
fileId = 5049;

ret = PGS_PC_GetReference(fileId, &version, my_HDF_file);

if (ret == PGS_S_SUCCESS)
{
    sdid1=SDstart(my_HDF_file, DFACC_CREATE);
}

printf("After SDstart sdid1 is %d\n",sdid1);

```

```

/***** write INVENTORYMETADATA to HDF4 file *****/
ret=PGS_MET_Write(handles[INVENTORYMETADATA],"coremetadata",sdid1);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF4 write failed\n");
    }
}

/***** write ARCHIVEDMETADAT to HDF4 file *****/
ret=PGS_MET_Write(handles[ARCHIVEDMETADATA],"archivemetadata",sdid1);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF4 write failed\n");
    }
}

/***** write to non-HDF file *****/

fileId = 5804;
printf("non-hdf file to be written has fileId %d\n", fileId);
ret=PGS_MET_Write(handles[ODL_IN_MEMORY],NULL,fileId);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("ASCII write failed\n");
    }
}

/***** write to default non-HDF file *****/

ret=PGS_MET_Write(handles[ODL_IN_MEMORY], NULL, NULL);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("ASCII write failed\n");
    }
}

```

```

/* write metadata to HDF5 file */
version =1;
fileId = 5059;

ret = PGS_PC_GetReference(fileId, &version, my_HDF5_file);

if (ret == PGS_S_SUCCESS)
{
    ret = PGS_MET_SDstart(my_HDF5_file, H5F_ACC_RDWR, &sdid5);
}

printf("After PGS_MET_SDstart sdid5 is %d\n",sdid5);

/***** write INVENTORYMETADATA to HDF5 file *****/

ret=PGS_MET_Write(handles[INVENTORYMETADATA],"coremetadata",sdid5);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF5 write failed\n");
    }
}

/***** write ARCHIVEDMETADAT to HDF5 file *****/

ret=PGS_MET_Write(handles[ARCHIVEDMETADATA],"archivemetadata",sdid5);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF5 write failed\n");
    }
}

SDend(sdidl);
(void) PGS_MET_SDend(sdid5);
PGS_MET_Remove();
free(informationname);

/* Initialize an ASCII metadata file */
fileId = 5801;

ret = PGS_MET_Init_NonMCF(fileId, mdHandles);
if (ret !=PGS_S_SUCCESS)
{
    printf("initialization failed for ASCII metadata file\n");
    return 0;
}
else
{
    printf("ret after PGS_MET_Init_NonMCF is %d\n",ret);
}
/* write metadata to HDF file */
version =1;
fileId = 5802;

```

```

ret = PGS_PC_GetReference(fileId, &version, my_HDF_file);

if (ret == PGS_S_SUCCESS)
{
    sdidl=SDstart(my_HDF_file, DFACC_CREATE);
}

printf("After SDstart sdidl is %d\n",sdidl);

/***** write INVENTORYMETADATA to HDF file *****/

ret=PGS_MET_Write(mdHandles[INVENTORYMETADATA],"coremetadata",sdidl);
printf("ret after PGS_MET_Write is %d\n",ret);

if(ret !=PGS_S_SUCCESS && ret != PGSMET_W_METADATA_NOT_SET)
{
    if (ret == PGSMET_E_MAND_NOT_SET)
    {
        printf("some mandatory parameters were not set\n");
    }
    else
    {
        printf("HDF write failed\n");
    }
}
SDend(sdidl);
PGS_MET_Remove();

printf("Complete...\n");
return 0;
}

```

The following is the PCFT file that is needed for the example above. The Path for the files needs to be changed to point where the input files are, and where the output files are to be created.

Note that for windows NT/98 one should use “\” instead of “/” in the directory structure.

```

#####
# The following IDs are defined in the TOOLKIT and they should not be changed
#####
10100|LogStatus|<TOOLKIT_MTD directory>/runtime/LogStatus
5000|configfile.dat|<TOOLKIT_MTD directory>/runtime/configfile.dat
10252|GetAttrtemp|<TOOLKIT_MTD directory>/test/test_MET/GetAttrtemp
10254|MCFWrite.temp|<TOOLKIT_MTD directory>/test/test_MET/MCFWrite.temp
10255|AsciiDump|<TOOLKIT_MTD directory>/test/test_MET/AsciiDump
10256|temporary.MCF|<TOOLKIT_MTD directory>/test/test_MET/temporary.MCF
10301|leapsec.dat|<TOOLKIT_MTD directory>/database/common/TD/leapsec.dat
10401|utcpole.dat|<TOOLKIT_MTD directory>/database/common/CSC/utcpole.dat
10402|earthfigure.dat|<TOOLKIT_MTD directory>/database/common/CSC/earthfigure.dat
10601|de200.eos|<TOOLKIT_MTD directory>/database/common/CBP/de200.eos
10801|sc_tags.dat|<TOOLKIT_MTD directory>/database/common/EPH/sc_tags.dat
10302|udunits.dat|<TOOLKIT_MTD directory>/database/common/CUC/udunits.dat
#####
# Logical IDs assigned for input/output files should be different from
# the IDs assigned above.

```

```

#####
10250|Carol_MCF_File|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/Carol_MCF_File
5039|MODIS_FILE.hdf|<TOOLKIT_MTD
directory>/test/test_MET/MET_TestData/MODIS_FILE.hdf.met
5049|MOD_FILE.hdf|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/MOD_FILE.hdf
5059|MODIS_FILE.h5|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/MODIS_FILE.h5
10251|data_dict|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/data_dict
5804|NAT_File|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/NAT_File
#####
# files to check PGS_MET_InitNonMCF function
#####
5801|MCFmorahan44|/<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/MCFmorahan44
5800|MOD10_L2_ASCII.met|/<TOOLKIT_MTD
directory>/test/test_MET/MET_TestData/MOD10_L2_ASCII.met
5802|MOD10_L2_HDF1.hdf|/<TOOLKIT_MTD
directory>/test/test_MET/MET_TestData/MOD10_L2_HDF1.hdf
#####
# End this table next two lines. Last line should be ?
#####
0|DUMMY|<TOOLKIT_MTD directory>/test/test_MET/MET_TestData/DUMMY
?

```

Appendix F. Config File Used by MET/TD Tools

During runtime some attributes are needed to be read (e.g. using `PGS_MET_GetConfigData`) from a file other than MCF. In SDP Toolkit, these attributes were available from the PCF file. For the Toolkit, these attributes will be available through a file that is referred to as Config file. This file has an entry in the PCFT file (see Appendix C) with a unique identifier (5000).

Each record contains an identifier (any arbitrary but unique number), attribute name, and attribute value(s). Note that the record fields are separated with a pipe token '|'.

Following is a sample Config file used for the MET and TD test drivers in the `$PGSHOME/test` directory.

```
#-----#
# The numbers 10120-10123 are used in TD test tools. They should not be changed #
#-----#
10123|TRMM UTCF value|0.0
10120|ADEOS-II s/c reference time|0.0
10121|ADEOS-II ground reference time|0.0
10122|ADEOS-II s/c clock period|0.0
#-----#
# MET test configuration parameters. The numbers 5991-5996 #
# can be any numbers, as long as they are not repeated #
#-----#
5991|TEST_PARM|1,2,3,4,5
5992|TEST_PARM_FLOAT|99.9
5993|TEST_PARM_STRING|SAT_0
5994|TEST_PARM_STRINGS|"SAT_1 is","a satellite"
5995|EOS_PLATFORM|OTD
5996|TEST_PARM_DOUBLE|0.0546781045
#-----#
# Last line should be a question mark (?). #
#-----#
?
```

This page is intentionally left blank

Appendix G. 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

AI&T	algorithm integration & test
API	application program interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
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
CSC	coordinate system conversion
CUC	CCSDS unsegmented time code
DAAC	distributed active archive center
DBMS	database management system
DCW	Digital Chart of the World
DEM	digital elevation model
DTM	digital terrain model
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
EOS	Earth Observing System

EOSAM	EOS AM Project (morning spacecraft series)
EOSDIS	Earth Observing System Data and Information System
EOSPM	EOS PM Project (afternoon spacecraft series)
EPH	ephemeris data access
ESDIS	Earth Science Data and Information
FOV	field of view
ftp	file transfer protocol
GAST	Greenwich apparent sidereal time
GCT	geo-coordinate transformation
GCTP	general cartographic transformation package
GMST	Greenwich mean sidereal time
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
HDF	hierarchical data format
http	hypertext transport protocol
IEEE	Institute of Electrical and Electronic Engineers
IERS	International Earth Rotation Service
IP	Internet protocol
JPL	Jet Propulsion Laboratory
LaRC	Langley Research Center
MCF	metadata configuration file
MEM	memory management
MET	metadata
MODIS	Moderate-Resolution Imaging Spectroradiometer
NASA	National Aeronautics and Space Administration
netCDF	network common data format
NGDC	National Geophysical Data Center
NMC	National Meteorological Center (NOAA)
ODL	object description language
PC	process control

PCF	process control file
PCFT	Private Customized File Table
PDPS	planning & data production system
PDS	production data set
PGE	product generation executive (formerly product generation executable)
PGS	Product Generation System
PGSTK	Product Generation System Toolkit
POSIX	Portable Operating System Interface for Computer Environments
SCF	Science Computing Facility
SDP	science data production
SDPF	science data processing facility
SDPS	Science Data Processing Segment (ECS)
SFDU	standard formatted data unit
SGI	Silicon Graphics Incorporated
SMF	status message file
SPCS	State Plane Coordinates Spheroid
SSM/I	Special Sensor for Microwave/Imaging
TAI	International Atomic Time
TD	time date conversion
TDB	Barycentric Dynamical Time
TDT	Terrestrial Dynamical Time
TRMM	Tropical Rainfall Measuring Mission (joint US – Japan)
UARS	Upper Atmosphere Research Satellite
UCAR	University Corporation for Atmospheric Research
URL	universal reference locator
UT	universal time
UTC	Coordinated Universal Time
UTCF	universal time correlation factor
UTM	universal transverse mercator
WWW	World Wide Web

This page intentionally left blank.