CERES Software Bulletin 97-03

February 26, 1997

How to Write Metadata to a Granule using the Toolkit

Alice Fan (t.f.fan@larc.nasa.gov)
Joe Stassi (j.c.stassi@larc.nasa.gov)

1.0 Purpose

This bulletin describes the tools and procedures needed to write metadata information during PGE processing. Refer to CERES Software Bulletin 97-02 for general background information on metadata.

2.0 ECS Metadata Approach

During PGE processing, each output file requires a Metadata Configuration File (MCF). The MCFs list the inventory and archive metadata attributes associated with the outputs; the inventory master group should be listed first. The developers are responsible for supplying a model MCF for each of their output file types. Two sample MCFs are mentioned within this bulletin: (1) MCF1 is associated with an HDF output file and (2) MCF2 is associated with a nonHDF output file.

Hierarchical Data Format (HDF) is the science data format selected by NASA as the baseline standard for ECS. ECS has chosen the attributes feature of HDF as the mechanism for writing metadata to HDF data products. Structural, inventory, and archive metadata will be written to the HDF global attributes section of the file in Object Description Language (ODL), a human-readable format. The Inventory metadata should be written with the HDF attribute name "coremetadata," and the archive metadata with the HDF attribute name "archivemetadata." Structural metadata apply only to output files in HDF-EOS format and are automatically handled by the HDF-EOS tools. When the toolkit writes inventory metadata to the HDF output file, it also writes them to a database load file. The database load file is an ASCII file used to populate the inventory database of the Science Data Server (SDSRV) at the completion of execution.

For nonHDF output files, the toolkit routines which write metadata must be called to send inventory metadata information to the inventory database; however, no metadata is written to the output file as a result of these procedures. For CERES, in order to provide a minimal amount of metadata information on binary output files, we will provide a CERES library routine to write a header record and another one to read the header record. The details of these procedures are still being developed and are not dealt with in this bulletin.

3.0 Writing Metadata from Fortran 90 Code

This bulletin is directed to users who will be writing metadata from Fortran 90 programs, and therefore all the examples are Fortran 90. In this section, the tools are listed along with some gen-

eral requirements for their use. Starting in Section 4.0, a detailed explanation of each tool is given along with an example showing its use. Appendix A shows an example of a complete program which writes metadata. The output from the program is shown in Appendix B.

3.1 Metadata Tools

There are seven Metadata tools provided by ECS in the toolkit library. These library routines are called from within PGE source code to read and write metadata information. The seven tools are listed below, along with a brief description of each one.

- **PGS_MET_Init( ):** Read MCF file into memory, assign values specified in MCF.
- **PGS_MET_SetAttr( )*:** Set value of PGE generated metadata.
- **PGS_MET_GetConfigData( )*:** Read runtime parameters from PCF using parameter name.
- **PGS_MET_GetSetAttr( )*:** Get from memory the metadata values set by Init or SetAttr.
- **PGS_MET_Write( ):** Write metadata to HDF file and write database load file.
- **PGS_MET_Remove( ):** Free up memory allocated by the ODL libraries.
- **PGS_MET_GetPCAttr( )*:** Read metadata from HDF file.

3.2 Fortran Version of Metadata Tools

Pay special attention to the four functions above which are marked with a *. The Fortran version of these functions 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 (datetime is stored as a string value), _I for integer and unsignedint 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 unsignedint values
- PGS_MET_SetAttr_D( ) to set real or double values

3.3 Calling Sequence

The metadata tools must be called in a certain general order within the PGE source code: (1) initialization, (2) setting metadata attribute values in memory and retrieving the values from memory, (3) writing metadata to output, and (4) freeing memory.

3.4 Fortran INCLUDE files

The Fortran toolkit routines require the following toolkit INCLUDE files.

include “PGS_MET.f”
include “PGS_MET_13.f”
include “PGS_SMF.f”

3.5 Required Scratch File Entries in the PCF

In the PCF, two scratch files with Logical ID 10252 and 10254 are required by the metadata tools.

a) Logical ID 10252 is used to dump metadata from the HDF attributes and is used by the PGS_MET_GetPCAttr( ) function. The default name for the file is GetAttr.temp.

b) Logical ID 10254 is similarly used to dump the MCF prior to writing to the HDF file. The default file name is MCFWrite.temp.

10252|AttrGet.temp|.||||1

10254|WriteMCF.temp|.||||1

These two logical IDs must be in the PRODUCT INPUT FILES section of the PCF. Metadata cannot be retrieved without the first file. They cannot be written without the second.

3.6 Checking the Result after a Function Call

After each function call, the result value returned by the function should be compared against the PARAMETER value PGS_S_SUCCESS (from the PGS_SMF.f include file). See the Toolkit User's Guide for a complete list of possible error conditions.

4.0 Initializing the MCF(s)

The toolkit function PGS_MET_Init( ) initializes an MCF. This must be done before metadata values can be set and written.

During initialization, the MCF is opened and read, and space is allocated in memory for the metadata attributes listed in the MCF. If a metadata attribute has its value defined in the MCF (Data_Location = "MCF"), then this value is set in memory during initialization.

If multiple MCFs are used during PGE processing, then the PGS_MET_Init( ) function must be called for each MCF. Currently, there is a limit of 20 MCFs which can be open at one time.

4.1 PGS_MET_Init( ): General Function Format

The general format of the PGS_MET_Init( ) function is

```
function PGS_MET_Init(&
    fileID, &       ! input parameter
    mdHandles)  ! output parameter
```

where

a) fileID (integer) is the logical ID in the PCF of the MCF being initialized.

b) mdHandles (character*) is the master group array containing handles to the master groups contained in the initialized MCF (see Section 4.3 for more information on master group arrays).

4.2 Using the PGS_MET_Init( ) Function

The following statements, extracted from the sample program in Appendix A, initialize the first MCF used by the program.

```
integer :: PGS_MET_Init
integer :: result
integer :: MCF1_ID
character (len=PGSd_MET_GROUP_NAME_L) :: &
    MasterGroup_MCF1(PGSd_MET_NUM_OF_GROUPS)

result = PGS_MET_Init(MCF1_ID, MasterGroup_MCF1)
```

4.3 Master Group Arrays

A master group array is a set of handles which point to the metadata master groups associated with an MCF. If multiple MCFs are used during PGE processing, then each MCF must have its own master group array.

Notes:

a) In the code, the master group array must be declared with 20 elements, 104 characters long. The toolkit INCLUDE file "PGS_MET.f" contains two PARAMETER values which may be used for the size and length of master group array declarations: PGSd_MET_NUM_OF_GROUPS = 20, and PGSd_MET_GROUP_NAME_L = 104.

b) The values in the master group array are set during initialization by the PGS_MET_Init( ) function. These values are needed as input for other toolkit metadata functions. Currently, only 3 elements of the array are used. The first element points to the root of the master group tree. The second element points to the inventory master group, and the third element points to the archive master group. If the MCF contains only inventory metadata, then only the first two elements of the array will be initialized.

c) In the example in Section 4.2, the array MasterGroup_MCF1 received the following values during initialization:
   • MasterGroup_MCF1(1)="MCF#0"
   • MasterGroup_MCF1(2)="INVENTORYMETADATA#0"
   • MasterGroup_MCF1(3)="ARCHIVEMETADATA#0"
   The "#0" indicates that MCF1_ID was the first MCF initialized in the code. If a second MCF is initialized, the values in its master group array will be given "#1", a third MCF would use "#2", and so forth.

5.0 Setting Metadata Values

The toolkit function PGS_MET_SetAttr( ) is called to set in memory the value of a metadata attribute. The attribute must have *Data_Location = {"PGE" or "PCF"}* in the MCF.

5.1 PGS_MET_SetAttr( ): General Function Format

The general format of the PGS_MET_SetAttr( ) function is

```
function PGS_MET_SetAttr(&
    mdHandle, attrName, attrValue)  ! input parameters
```

where

a) mdHandle (character*) is the handle pointing to the metadata master group which contains the attribute.

b) attrName (character*) is the metadata attribute name.
c) attrValue ('user defined') is the metadata attribute value.

5.2 Setting Individual Value Attributes

The sample code below sets a value for the "RangeBeginningTime" attribute in the inventory master group of the first MCF. Note that since the attribute value type is string, the PGS_MET_SetAttr_S( ) function is called.

```
integer :: PGS_MET_SetAttr_S
integer, parameter :: INVENTORY_MD = 2
character (len=10) :: datestring
character (len=PGSd_MET_GROUP_NAME_L) :: mdHandle2

datestring = ... ! (user defined)
mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
result = PGS_MET_SetAttr_S(mdHandle2, "RangeBeginningDate", datestring)
```

In the following code, a value for the "Browsesize" attribute in the archive master group of the first MCF is set. Since the attribute value type is real, the PGS_MET_SetAttr_D( ) function is called.

```
integer :: PGS_MET_SetAttr_D
integer, parameter :: ARCHIVE_MD = 3
real :: dval
character (len=PGSd_MET_GROUP_NAME_L) :: mdHandle3

dval = ... ! (user defined)
mdHandle3 = MasterGroup_MCF1(ARCHIVE_MD)
result = PGS_MET_SetAttr_D(mdHandle3, "Browsesize", dval)
```

5.3 Setting Multi-value (Array) Attributes

When setting a multi-value (array) attribute, all the values of the attribute must be set with a single statement. Array elements cannot be set individually.

The following statements set three values for the multi-value attribute "ZoneIdentifier" in the inventory master group of the first MCF.

```
integer :: PGS_MET_SetAttr_I
integer :: ivals(6)
ivals(1) = ... ! (user defined)
ivals(2) = ... ! (user defined)
ivals(3) = ... ! (user defined)
ivals(4) = PGSd_MET_INT_MAX

mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
result = PGS_MET_SetAttr_I(mdHandle2, "ZoneIdentifier", ivals)
```

The dimension of the local array, ivals, should equal the Num_Val value for ZoneIdentifier in the MCF. If the number of values being set is less than the dimension, then the first unused

element of the array should be set to PGSd_MET_INT_MAX, PGSd_MET_STR_END, or PGSd_MET_DBL_MAX, depending on the data type of the array.

5.4 Setting Multiple Class Attributes

In contrast to multi-value objects, each value of a multiple class object must be set separately.

The following code shows how the multiple-class objects, ParameterName and ParameterValue, in the InformationContentContainer Group can be used to set values for two user-specified attributes: "ProductionDateTime" and "PlatformShortName."

```
USE ceres_time, only : HeaderTime
integer :: PGS_MET_SetAttr_S
character (len=30) :: DateTime

!*** Set "ProductionDateTime" attribute
DateTime = Header_Time()
mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
result =PGS_MET_SetAttr_S(mdHandle2, "ParameterName.1", "ProductionDateTime")
result = PGS_MET_SetAttr_S(mdHandle2, "ParameterValue.1", DateTime)

!*** Set "PlatformShortName" attribute
result =PGS_MET_SetAttr_S(mdHandle2, "ParameterName.2", "PlatformShortName")
result = PGS_MET_SetAttr_S(mdHandle2, "ParameterValue.2", "TRMM")
```

Since "ProductionDateTime" and "PlatformShortName" have not yet been identified as attributes within the MCF template, the only way to enter these values are as user-specified attributes.

Notes:

a) The different instances of the ParameterName and ParameterValue objects are distinguished by a decimal point after the name, followed by an integer counter.

b) The Header_Time( ) function is a CERES library routine in the ceres_time module. It returns current time in the following format: "1997-02-21T16:39:54.000000Z"

6.0 Retrieving Metadata Values from the PCF

At one time, it was thought that metadata values stored on the PCF would automatically be retrieved and set in memory during initialization, similar to how values stored on the MCF are handled. This was probably the original purpose of the *Data_Location = "PCF"* parameter in the MCF. This capability, however, was never developed, and the PGE software must explicitly retrieve the metadata attribute values from the PCF and then set the values using the PGS_MET_SetAttr( ) function.

The function which retrieves values from the PCF is the PGS_MET_GetConfigData( ) function.

6.1 PGS_MET_GetConfigData( ): General Function Format

The general format of the PGS_MET_GetConfigData( ) function is

```
function PGS_MET_GetConfigData(&
    attrName,  ! input parameter
```

attrValue) ! output parameter

where
a) attrName (character*) is the metadata attribute name.
b) attrValue ('user defined') is the metadata attribute value which is returned.

6.2 Using the PGS_MET_GetConfigData( ) Function

The statement below shows how the value for "YEAR" is retrieved from the PCF using the PGS_MET_GetConfigData( ) function. Once the value is retrieved, it must be set in memory using the PGS_MET_SetAttr( ) function explained in the previous sections.

integer :: PGS_MET_GetConfigData_I
integer :: yearfrompcf
result = ***PGS_MET_GetConfigData_I****("YEAR", yearfrompcf)*

Notes:
a) An entry similar to *110|YEAR|1989* must exist in the PCF. The logical ID (the first field) is not used and is irrelevant to this call.
b) The value of the first parameter of the function call, "YEAR", must match the 2nd field of the PCF entry.
c) The value field (the third field) in the PCF entry can be of type string, integer, unsignedint, double, or datetime. The second parameter of the function call, yearfrompcf, must be of the same type.
d) This function duplicates two other toolkit functions which retrieve runtime parameters from the PCF using the PCF logical ID: PGS_PC_GetConfigData( ) and PGS_PC_GetConfigDataCom( ). These other functions are addressed in the CERES software bulletin 95-16.

7.0 Retrieving Set Metadata Values from Memory

Sometimes it is necessary to retrieve a metadata attribute value which has been set in memory, particularly those values which are stored in the MCF and set during initialization. The PGS_MET_GetSetAttr( ) function provides this capability.

7.1 PGS_MET_GetSetAttr: General Function Format

The general format of the PGS_MET_GetSetAttr( ) function is

function PGS_MET_GetSetAttr(&
mdHandle, attrName, & ! input parameters
attrValue) ! output parameter

where
a) mdHandle (character*) is the handle pointing to the metadata master group which contains the attribute.
b) attrName (character*) is the metadata attribute name.
c) attrValue ('user defined') is the metadata attribute value which is returned.

7.2 Using the PGS_MET_GetSetAttr( ) Function

The following function call returns the values for ParameterName.1 and ParameterValue.1 from memory into the array svalsRet.

integer :: PGS_MET_GetSetAttr_S
character (len=8) :: svalsRet(2)

mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
*result = **PGS_MET_GetSetAttr_S**(mdHandle2, "ParameterName.1", svalsRet(1))*
*result = **PGS_MET_GetSetAttr_S**(mdHandle2, "ParameterValue.1", svalsRet(2))*

8.0 Writing Metadata to Output

The toolkit function PGS_MET_Write( ) writes metadata which has been set in memory.

The use of PGS_MET_Write( ) and the results from it vary depending on whether the function is called to write metadata for an HDF output file or for a nonHDF output file. Each case is discussed in more detail in the following sections, but here is a summary of the differences.

1. When writing metadata for an HDF output file, the PGS_MET_Write( ) function must be called separately for the inventory and archive metadata master groups. For a nonHDF output file, the function is called only once using the root of the master group tree.
2. For HDF output files, the metadata are written directly to the output file, and the inventory metadata are also automatically written to a database load file. For nonHDF output files, only the database load file gets written. No metadata are written to the output file with this function. For more information on database load files, see Section 8.2.

Since metadata is written by master groups, all metadata attributes belonging to a master group of an MCF must have their values set before that master group is written.

8.1 PGS_MET_Write( ): General Function Format

The general format of the PGS_MET_Write function is

function PGS_MET_Write(&
*mdHandle, hdfAttrName, hdfFileId) !*** input parameters*

where

a) mdHandle (character*) is the handle of the metadata master group being written.

b) hdfAttrName (character*) is the HDF attribute name to be associated with the metadata master group being written (use "coremetadata" for the inventory master group and "archivemetadata" for the archive master group). This parameter does not apply when writing metadata for a nonHDF file.

c) hdfFileId (integer) is the output file id. For HDF output files, this is the HDF ID returned by the HDF file open routine. For nonHDF output files, this is the PCF logical ID of the database load file associated with the nonHDF output.

8.2 Database Load Files

A database load file is an ASCII file created during PGE processing. It is used to transport the inventory metadata information about an output product to the inventory database after PGE

execution is complete. Each PGE output file must have its own database load file. Database load files for nonHDF outputs are handled differently than load files for HDF outputs.

Each nonHDF output file must have a separate entry for its database load file in the PRODUCT OUTPUT FILES section of the PCF. The database load file must be named xxxx.met, where xxxx is the associated output product file name.

For HDF output files, no PCF entry is necessary for the database load files. The load files are created automatically by the metadata tools, and their names are derived from the corresponding output product file names, using the same naming scheme described above.

8.3 Writing Metadata Attributes for HDF Output

The following statements write the metadata values for the HDF output file associated with the first MCF.

```
integer :: PGS_MET_Write

CALL GetFileName(HDF_ID, version=1, FileName=hdffilename, status=getstatus)
sdid = sfstart(hdffilename, access)

mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
mdHandle3 = MasterGroup_MCF1(ARCHIVE_MD)

result=PGS_MET_Write(mdHandle2, "coremetadata", sdid)
result=PGS_MET_Write(mdHandle3, "archivemetadata", sdid)
```

Notes:

a) GetFileName is a CERES library wrapper routine for the PGS_PC_GetReference toolkit function. GetFileName is discussed in CERES Software Bulletin 95-16.
b) HDF_ID is the HDF file logical ID from the PCF.
c) The HDF library function sfstart returns a file handle (unit number). This value is stored in the variable sdid.
d) For HDF output files, it is not possible to write out both metadata master groups with a single call under Toolkit 5.1.1.
e) "coremetadata" and "archivemetadata" are HDF global attribute names for the inventory and archive metadata, respectively.
f) The second call to PGS_MET_Write( ) is not necessary if there is no archive metadata.

8.4 Writing Metadata Attributes for nonHDF Output

The following statements write the metadata associated with a nonHDF output file using the second MCF. Note that the metadata values are not physically written on the nonHDF file, but are written to the database load file.

```
integer :: PGS_MET_Write
integer, parameter :: ODL_IN_MEMORY = 1
character (len =1) :: dummyStr = " "
integer, parameter :: DBload_ID  = 255  ! Logical ID of database load file

mdHandle1 = MasterGroup_MCF2(ODL_IN_MEMORY)
result = PGS_MET_Write(mdHandle1, dummyStr, DBload_ID)
```

Notes:
a) ODL_IN_MEMORY=1 is the index into the master group array pointing to the root of the ODL master group tree.
b) dummyStr is simply a place holder since hdfAttrName is not required for nonHDF files.
c) DBload_ID is the PCF logical ID of the database load file associated with the nonHDF output file. The load file must be named xxxx.met in the PCF, where xxxx is the name of the nonHDF output file.
d) For nonHDF output files under Toolkit 5.1.1, all master groups are written collectively.

9.0 Freeing Memory

The PGS_MET_Remove( ) function frees up memory which has been allocated to initialized MCFs.

9.1 PGS_MET_Remove( ): General Function Format

The general format of the PGS_MET_Remove( ) function is

function PGS_MET_Remove( )

There are no arguments to this function call. It acts on memory allocated to all initialized MCFs. Therefore, it should not be called until all metadata from all initialized MCFs have been written.

9.2 Using the PGS_MET_Remove( ) Function

The following statements free up memory after all metadata have been written.

integer :: PGS_MET_Remove

*result =**PGS_MET_Remove**( )*

10.0 Reading Metadata from an Existing HDF File

Metadata can be read from an existing HDF file using the PGS_MET_GetPCAttr( ) function. This is an independent function, i.e. it does not require that other functions be called before or after it.

10.1 PGS_MET_GetPCAttr( ): General Function Format

The general format of the PGS_MET_GetPCAttr( ) function is

```
function PGS_MET_GetPCAttr(&
    fileId, version, hdfAttrName, parmName, &  ! input parameters
    parmValue)                                 ! output parameter
```

where
a) fileId (character*) is the PCF logical ID for the HDF file being read.
b) version (integer) is the PCF version number of the HDF file.
c) hdfAttrName (character*) is the HDF attribute name of the master group.
d) parmName (character*) is the metadata attribute name.

e) parmValue ('user define') is the metadata attribute value which is returned.

10.2 Using the PGS_Met_GetPCAttr( ) Function

The following statements will retrieve the RangeBeginningDateTime attribute from the inventory metadata of an existing HDF file into the datetimeRet1 variable.

```
integer :: PGS_MET_GetPCAttr
integer :: pcfVersionNum
character (len=10) :: datetimeRet1

pcfVersionNum = 1  !** This example chooses the first instance of the file in the PCF
result = PGS_MET_GetPCAttr_S(OLD_HDF_ID, pcfVersionNum, "coremetadata", &
    "RangeBeginningDateTime", datetimeRet1)
```

Notes:
a) OLD_HDF_ID is the logical ID for the HDF file specified in the PCF.
b) The HDF attribute name "coremetadata" (or "archivemetadata") must have been used to create the metadata in order to retrieve the metadata from the HDF file using the same name.

11.0 Outcome of Running the Metadata Tools

Appendix D shows the result of the WRITE function; the inventory metadata which is written to the database load file. On an HDF-EOS data file, metadata will be written in the following order: structure, inventory, and archive.

If a metadata attribute has not been set, and it has MANDATORY = "TRUE" in the MCF, then the function result value will equal PGSMET_E_MAND_NOT_SET, a PARAMETER value found in the PGS_MET_13.f include file. This condition is fatal and will keep the inventory metadata information from being loaded to the Science Data Server database.

The value for unset attributes are set as follows:

Data_location	Value
PGE	"NOT SET"
PCF	"NOT FOUND"
MCF	"NOT SUPPLIED"

If a metadata attribute has not been set, and it has MANDATORY = "FALSE" in the MCF, a message PGSMET_W_METADATA_NOT_SET is issued and the specific attribute will not appear in the metadata or the ASCII file. For example the attribute LocalityValue in the MCF file of Appendix C is not shown in the output file of Appendix D.

For attributes of type datetime, two additional attributes are produced. For example the RangeBeginningDateTime attribute in the MCF of Appendix C generates two attributes, RangeBeginningDate and RangeBeginningTime, in the metadata output file of Appendix D.

Appendix A: Sample f90 Program for Writing Metadata

```
PROGRAM test_meta
  !ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
  ! This program was tested with two MCFs :
  ! The first MCF has two master groups - inventory and archive
  ! The second MCF has only one master group - inventory
  !ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

  USE ceres_time, ONLY : Header_time
  USE f90_kind,   ONLY : real8
  USE io,         ONLY : OpenFile, CloseFile
  USE pcf,        ONLY : GetFileName

  IMPLICIT NONE
  INCLUDE 'PGS_MET.f'
  INCLUDE 'PGS_MET_13.f'
  INCLUDE 'PGS_SMF.f'

  !-------------------------------------------------------------------------!
  ! The following declaration for Metadata handle                           !
  ! "character*104  mastergroups(20)" is required.                          !
  ! The handles get the following values once PGS_MET_Init( ) is called.    !
  !             MasterGroups(1)="MCF"                                       !
  !             MasterGroups(2)="INVENTORYMETADATA"                         !
  !             MasterGroups(3)="ARCHIVEMETADATA"                           !
  !                                                                         !
  ! TK defines PGSd_MET_GROUP_NAME_L = 104   PGSd_MET_NUM_OF_GROUPS = 20    !
  !-------------------------------------------------------------------------!
  CHARACTER (LEN=PGSd_MET_GROUP_NAME_L) :: &
       MasterGroup_MCF1(PGSd_MET_NUM_OF_GROUPS)
  CHARACTER (LEN=PGSd_MET_GROUP_NAME_L) :: mdHandle1, mdHandle2, mdHandle3

  !-------------------------------------------------------------------------!
  !  The following three lines are recommended and are used to refer to     !
  !  each group                                                             !
  !-------------------------------------------------------------------------!
  INTEGER, PARAMETER :: ODL_IN_MEMORY = 1     ! tree root
  INTEGER, PARAMETER :: INVENTORY_MD  = 2     ! leaf
  INTEGER, PARAMETER :: ARCHIVE_MD    = 3     ! leaf

  !-------------------------------------------------------------------------!
  ! This test program assumes the following logical IDs are in the          !
  ! PRODUCT INPUT FILES section of the PCF.                                 !
  !                                                                         !
  ! The logical IDs except 10252 and 10254 are user's choice.               !
  ! The logical ID 10252 and 10254 are required by the toolkit operation.   !
  !                                                                         !
  ! The following file names and file paths are your choice.                !
  !           250|MCF1|/disk2/thunder/fan/Meta1||||1                        !
  !           251|MCF2|/disk2/thunder/fan/Meta1||||1                        !
  !           100|xxxxxx|/disk2/thunder/fan/Meta1|||nonHDF|1                !
  !           200|xxxxxx|/disk2/thunder/fan/Meta1|||HDFfile|1               !
```

```
!           10252|GetAttr.temp|/disk2/thunder/fan/Meta1||||1            !
!           10254|MCFWrite.temp|/disk2/thunder/fan/Meta1||||1           !
!                                                                       !
!    The logical IDs, 250 and 251, point to the MCF files.              !
!    The logical IDs, 100 and 200, point to an existing HDF or          !
!    nonHDF file.  They are for testing the                             !
!    PGS_MET_GetPCAttr( ) function to read metadata from                !
!                                                                       !
! This test program assumes the following logical IDs are in the        !
! PRODUCT OUTPUT FILES section of the PCF. They are used for writing    !
! metadata to an HDF file and a nonHDF file.                            !
!                                                                       !
!           253|HDFfile|/disk2/thunder/fan/Meta1||||1                   !
!           255|nonHDF|/disk2/thunder/fan/Meta1||||1                    !
!                                                                       !
!-----------------------------------------------------------------------!
INTEGER, PARAMETER :: MCF1_ID    = 250  ! Logical ID, first MCF (input)
INTEGER, PARAMETER :: OLD_HDF_ID = 200  ! Logical ID, HDF file (input)
INTEGER, PARAMETER :: HDF_ID     = 253  ! Logical ID, HDF file (output)
INTEGER, PARAMETER :: DBload_ID  = 255  ! Logical ID, database load file for
                                        ! the non-HDF output

!***  The seven Metadata Tools
INTEGER :: PGS_MET_Init
INTEGER :: PGS_MET_SetAttr_S, PGS_MET_SetAttr_I, PGS_MET_SetAttr_D
INTEGER :: PGS_MET_GetSetAttr_S, PGS_MET_GetSetAttr_I, PGS_MET_GetSetAttr_D
INTEGER :: PGS_MET_GetPCAttr_S, PGS_MET_GetPCAttr_I,PGS_MET_GetPCAttr_D
INTEGER :: PGS_MET_Write
INTEGER :: PGS_MET_GetConfigData_S, PGS_MET_GetConfigData_I
INTEGER :: PGS_MET_Remove

CHARACTER (LEN=20)  :: svalsRet(3)
CHARACTER (LEN=30)  :: Datetime, Longname
CHARACTER (LEN=10)  :: DateRet, TimeRet, DateRet1, TimeRet1
CHARACTER (LEN=10)  :: Datestring, Timestring
CHARACTER (LEN=200) :: hdffilename
CHARACTER (LEN=1)   :: dummyStr

INTEGER :: result, getstatus
INTEGER :: access, i
INTEGER :: ival, ivalRet
INTEGER :: ivals(6), ivalsRet(6)
INTEGER :: sdid, hdfReturn, sfstart, sfend, yearfrompcf
REAL(real8) :: dval, dvalRet

!----------------!
!   INITIALIZE   !
!----------------!
result = PGS_MET_Init(MCF1_ID, MasterGroup_MCF1)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "Initialization for MCF1_FILE error"
ELSE
   PRINT*, "Initialization for MCF1_FILE works"
END IF
```

```
!*** set handles for first MCF
mdHandle1 = MasterGroup_MCF1(ODL_IN_MEMORY)
mdHandle2 = MasterGroup_MCF1(INVENTORY_MD)
mdHandle3 = MasterGroup_MCF1(ARCHIVE_MD)

!---------------------------!
!   SET INTEGER attribute   !
!---------------------------!
ival = 30
result =PGS_MET_SetAttr_I(mdHandle2, "QAPercentOutofBoundsData", ival)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for OutofBounds =", result
END IF

!---------------------------!
!   SET String attribute   !
!---------------------------!
Datetime = Header_Time( )            ! get the current time in UTC format
Datestring = Datetime(1:10)
Timestring = Datetime(12:26)

result = PGS_MET_SetAttr_S(mdHandle2, "RangeBeginningDate", Datestring)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for RangeBeginningDate     =", result
END IF

result = PGS_MET_SetAttr_S(mdHandle2, "RangeBeginningTime", Timestring)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for RangeBeginningTime     =", result
END IF

!---------------------------!
!   SET double attribute   !
!---------------------------!
dval = 220.2
result = PGS_MET_SetAttr_D(mdHandle2, "EastBoundingCoordinate", dval)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for EastBound =", result
END IF

!-------------------------------------------------------------------------!
!   SET Array attribute                                                   !
!                                                                         !
!       The whole array needs to be set all at once. It shall not be      !
!       set one by one using the following two lines else the result      !
!       will be incorrect.                                                !
!                                                                         !
!       result =PGS_MET_SetAttr_I(MasterGroup_MCF1(INVENTORY_MD), &       !
!            "ZoneIdentifier,ivals(1))                                    !
!       result =PGS_MET_SetAttr_I(MasterGroup_MCF1(INVENTORY_MD), &       !
!            "ZoneIdentifier,ivals(2))                                    !
!                                                                         !
!       You need to put PGSd_MET_INT_MAX, PGSd_MET_STR_END, or            !
```

```
!       PGSd_MET_DBL_MAX respectively at the last element, if your     !
!       array is not fully defined.                                    !
!--------------------------------------------------------------------!
DO i = 1,3
   ivals(i) = i*10
END DO

ivals(4) = PGSd_MET_INT_MAX
result = PGS_MET_SetAttr_I(mdHandle2, "ZoneIdentifier", ivals)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for ZoneIdentifier = ", result
END IF

!---------------------------------------!
!   SET multiplicity attribute          !
!       It needs to be set one by one   !
!---------------------------------------!
result = PGS_MET_SetAttr_S(mdHandle2, "ParameterName.1", &
    "ProductionDateTime")
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for ParameterName.1, result = ", result
END IF
result = PGS_MET_SetAttr_S(mdHandle2, "ParameterValue.1", DateTime)
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for ParameterValue.1, result = ", result
END IF

result = PGS_MET_SetAttr_S(mdHandle2, "ParameterName.2", &
    "PlatformShortName")
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for ParameterName.2, result = ", result
END IF
result = PGS_MET_SetAttr_S(mdHandle2, "ParameterValue.2","TRMM")
IF (result /= PGS_S_SUCCESS) THEN
   PRINT*, "SET ERROR for ParameterValue.1, result = ", result
END IF

!-----------------------------------------------!
!   SET an object in the Archive Metadata group  !
!-----------------------------------------------!
dval = 321.123
result = PGS_MET_SetAttr_D(mdHandle3, "BrowseSize", dval)

!-------------------------------!
!   GET a Set attribute value   !
!-------------------------------!
result = PGS_MET_GetSetAttr_I(mdHandle2, "QAPercentOutofBoundsData", &
    ivalRet)
PRINT*, "From GetSetAttr, QAPercentOutofBoundsData    = ", ivalRet

result = PGS_MET_GetSetAttr_S(mdHandle2, "RangeBeginningDate", dateRet)
PRINT*, "From GetSetAttr, RangeBeginningData          = ", dateRet

result = PGS_MET_GetSetAttr_S(mdHandle2, "RangeBeginningTime", TimeRet)
```

```
PRINT*, "From GetSetAttr, RangeBeginningTime          = ", TimeRet

result = PGS_MET_GetSetAttr_D(mdHandle2, "EastBoundingCoordinate", dvalRet)
PRINT*, "From GetSetAttr, EastBoundingCoordinate      = ", dvalRet

result = PGS_MET_GetSetAttr_I(mdHandle2, "ZoneIdentifier", ivalsRet)
PRINT*, "From GetSetAttr, ZoneIdentifier              = ",     &
     ivalsRet(1), ivalsRet(2), ivalsRet(3)

result = PGS_MET_GetSetAttr_S(mdHandle2, "ParameterName.1", svalsRet(1))
result = PGS_MET_GetSetAttr_S(mdHandle2, "ParameterValue.1", svalsRet(2))
PRINT*, "From GetSetAttr, ParameterName.1 and Value.1 = ",    &
     svalsRet(1), svalsRet(2)

result = PGS_MET_GetSetAttr_S(mdHandle2, "ParameterName.2", svalsRet(1))
result = PGS_MET_GetSetAttr_S(mdHandle2, "ParameterValue.2", svalsRet(2))
PRINT*, "From GetSetAttr, ParameterName.2 and Value.2 = ",    &
     svalsRet(1), svalsRet(2)

result = PGS_MET_GetSetAttr_D(mdHandle3, "BrowseSize",dvalRet)
PRINT*, "From GetSetAttr, BrowseSize                  = ", dvalRet
PRINT*, " "

!----------------------!
!   OPEN an HDF file   !
!----------------------!
CALL GetFileName (HDF_ID, version=1, FileName=hdffilename, &
     status=getstatus)

IF (getstatus == 0) THEN
   PRINT*, "get HDF file name =", hdffilename
ELSE
   PRINT*, "get HDF file name fail"
END IF

access = 4                !     (DFACC_CREATE)
sdid = sfstart(hdffilename, access)
IF (sdid == -1) THEN
   PRINT*, "Failed to open the HDF file"
ELSE
   PRINT*, "sdid = ", sdid
END IF

!==================================!
!   WRITE Metadata for an HDF File   !
!==================================!
!-------------------------------------------------------------------------!
!   Here's how the metadata would be written for an HDF file.             !
!   WRITE to an HDF file one master group (INVENTORY, ARCHIVE) at a time   !
!          - It automatically writes the inventory group to an ASCII file, !
!            when the inventory group is written to an HDF file.          !
!                                                                         !
!   It is recommended to write the inventory master group first and       !
!   archive master group second.                                          !
```

```
!                                                                          !
!   The attribute names "coremetadata" and "archivemetadata" are not       !
!   supposed to be changed.                                                !
!--------------------------------------------------------------------------!
result = PGS_MET_Write(mdHandle2, "coremetadata", sdid)
PRINT*, "WRITE INVENTORY  group to an HDF, result =", result

result = PGS_MET_Write(mdHandle3, "archivemetadata", sdid)
PRINT*, "WRITE ARCHIVE    group to an HDF, result =", result

!-------------------------!
!   CLOSE the HDF file    !
!-------------------------!
hdfReturn = sfend(sdid)
PRINT*, "Close HDF file hdfReturn=", hdfReturn

!=====================================!
!   WRITE Metadata for an nonHDF File   !
!=====================================!
!--------------------------------------------------------------------------!
!   Here's how the metadata would be written for an HDF file.              !
!    WRITE to an nonHDF file of your own choice                            !
!          It writes out both inventory and archive master groups all      !
!          at once.  You can not write out only a selected group by using  !
!                                                                          !
!      result = PGS_MET_Write(MasterGroup_MCF1(INVENTORY_MD), dummyStr, &  !
!          DBload_ID)                                                      !
!          DBload_ID (logical ID) needs to be in PCF.                      !
!--------------------------------------------------------------------------!
result = PGS_MET_Write(mdHandle1, dummyStr, DBload_ID)
PRINT*, "WRITE every group to a nonHDF file, result = ", result

!-------------------------------------!
!   Free up memory allocated by MCF    !
!-------------------------------------!
result =PGS_MET_Remove( )

!--------------------------------------------------------------------------!
!  READ metadata from an existing HDF file                                 !
!      OLD_HDF_ID(logical ID) needs to be specified in the PCF             !
!      "coremetadata" and "archivemetadata" HDF attribute names must have  !
!      been set when the metadata was written.                             !
!--------------------------------------------------------------------------!
result = PGS_MET_GetPCAttr_S(OLD_HDF_ID, 1, "coremetadata", &
     "RangeBeginningDate", dateRet1)
PRINT*, "READ from an OLD HDF    file RangebeginningDate = ", dateRet1

result = PGS_MET_GetPCAttr_S(OLD_HDF_ID, 1, "coremetadata", &
     "RangeBeginningTime", TimeRet1)
PRINT*, "READ from an OLD HDF    file RangebeginningTime = ", TimeRet1

result = PGS_MET_GetPCAttr_D(OLD_HDF_ID, 1, "archivemetadata",&
     "Browsesize", dvalRet)
PRINT*, "READ from an OLD HDF    file BrowseSize = ",dvalRet
```

```
  PRINT*, " "

  !-----------------------------------------------------------------!
  !   Get RUN TIME Parameter from PCF file - assume the following   !
  !   two lines are in PCF                                          !
  !                                                                 !
  !       110|YEAR|1989                                            !
  !       120|LONGNAME|ssf_19861001_01                             !
  !                                                                 !
  !-----------------------------------------------------------------!
  result = PGS_MET_GetConfigData_I("YEAR",yearfrompcf)
  PRINT*, "READ from PCF   YEAR   = ", yearfrompcf

  result = PGS_MET_GetConfigData_s("LONGNAME",longname)
  PRINT*, "READ from PCF Longname = ", longname

END PROGRAM test_meta
```

Appendix B: Output from Sample f90 Program

```
Initialization for MCF1_FILE works

From GetSetAttr, QAPercentOutofBoundsData    =  30
From GetSetAttr, RangeBeginningData          = 1997-02-24
From GetSetAttr, RangeBeginningTime          = 17:51:47.0
From GetSetAttr, EastBoundingCoordinate      =    2.2019999700000000E+02
From GetSetAttr, ZoneIdentifier              =  10 20 30
From GetSetAttr, ParameterName.1 and Value.1 = ProductionDateTime
            1997-02-24T17:51:47.
From GetSetAttr, ParameterName.2 and Value.2 = PlatformShortName  TRMM
From GetSetAttr, BrowseSize                  =    3.2112298600000003E+02

getHDFfilename=./HDFfile
sdid =  393216
WRITE INVENTORY  group to an HDF, result = 110148
WRITE ARCHIVE    group to an HDF, result = 0
Close HDF file hdfReturn= 0

WRITE every group to a nonHDF file, result =  110148
READ from an OLD HDF    file RangebeginningDate = 1997-02-24
READ from an OLD HDF    file RangebeginningTime = 17:51:47.0
READ from an OLD HDF    file BrowseSize =    3.2112298600000003E+02

READ from PCF   YEAR   =  1989
READ from PCF Longname = SSF_19861001
0.0u 0.0s 0:00  0+0k 368k 4+16io 0pf+0w
```

* note that the return code of 110148 means some mandatory attributes were not set.

Appendix C: Sample MCF

```
GROUP = INVENTORYMETADATA
  GROUPTYPE = MASTERGROUP

      OBJECT = ShortName        /* Name of attribute */
          Data_Location     = "MCF"    /* Set in MCF */
          Value             = "COGGEOAA"   /* The value replaces xxxx */
          TYPE              = "STRING"
          NUM_VAL           = 1
          Mandatory         = "TRUE"
      END_OBJECT = ShortName

      OBJECT = VersionID
          Data_Location     = "MCF"
          Value             = "1"
          TYPE              = "STRING"
          NUM_VAL           = 1
          Mandatory         = "TRUE"
      END_OBJECT = VersionID

      OBJECT = SizeMBECSDataGranule
          Data_Location     = "PGE"
          TYPE              = "DOUBLE"
          NUM_VAL           = 1
          Mandatory         = "TRUE"
      END_OBJECT = SizeMBECSDataGranule

  GROUP = InputGranule

      OBJECT = InputPointer
          Data_Location     = "PGE"
          TYPE              = "STRING"
          NUM_VAL           = 1     /* Number of inputs to product. */
          Mandatory         = "TRUE"
      END_OBJECT = InputPointer

  END_GROUP = InputGranule

  GROUP = RangeDateTime

      /* Format of RangeBeginningDate is YYYY-MM-DD or YYYY-DDD */
      OBJECT = RangeBeginningDate
          Data_Location     = "PGE"
          TYPE              = "DATE"
          NUM_VAL           = 1
          Mandatory         = "TRUE"
      END_OBJECT = RangeBeginningDate

      /* Format of RangeBeginningTime is HH:MM:SS.SSSS... */
      OBJECT = RangeBeginningTime
          Data_Location     = "PGE"
          TYPE              = "TIME"
```

```
        NUM_VAL           = 1
        Mandatory         = "TRUE"
    END_OBJECT = RangeBeginningTime

    /* Format of RangeEndingDate is YYYY-MM-DD or YYYY-DDD */
    OBJECT = RangeEndingDate
        Data_Location    = "PGE"
        TYPE             = "DATE"
        NUM_VAL          = 1
        Mandatory        = "TRUE"
    END_OBJECT = RangeEndingDate

    /* Format of RangeEndingTime is HH:MM:SS.SSSS... */
    OBJECT = RangeEndingTime
        Data_Location     = "PGE"
        TYPE              = "TIME"
        NUM_VAL           = 1
        Mandatory         = "TRUE"
    END_OBJECT = RangeEndingTime

END_GROUP = RangeDateTime

GROUP = ZoneIdentifierClass

    OBJECT = ZoneIdentifier
        Data_Location = "PGE"
        TYPE = "INTEGER"
        NUM_VAL =6
        Mandatory = "FALSE"
    END_OBJECT = ZoneIdentifier

END_GROUP = ZoneIdentifierClass

GROUP = BoundingRectangle

    OBJECT = EastBoundingCoordinate
        Data_Location     = "PGE"
        TYPE              = "DOUBLE"
        NUM_VAL           = 1
        Mandatory         = "FALSE"
    END_OBJECT = EastBoundingCoordinate

    OBJECT = WestBoundingCoordinate
        Data_Location     = "PGE"
        TYPE              = "DOUBLE"
        NUM_VAL           =  1
        Mandatory         = "FALSE"
    END_OBJECT = WestBoundingCoordinate

    OBJECT = NorthBoundingCoordinate
        Data_Location     = "PGE"
        TYPE              = "DOUBLE"
        NUM_VAL           = 1
        Mandatory         = "FALSE"
```

```
    END_OBJECT = NorthBoundingCoordinate

    OBJECT = SouthBoundingCoordinate
        Data_Location     = "PGE"
        TYPE              = "DOUBLE"
        NUM_VAL           = 1
        Mandatory         = "FALSE"
    END_OBJECT = SouthBoundingCoordinate

END_GROUP = BoundingRectangle

GROUP = QAStats

    OBJECT = QAPercentInterpolatedData
         Data_Location    = "PGE"
         TYPE             = "INTEGER"
         NUM_VAL          = 1
         Mandatory        = "FALSE"
    END_OBJECT = QAPercentInterpolatedData

    OBJECT =  QAPercentOutofBoundsData
         Data_Location    = "PGE"
         TYPE             = "INTEGER"
         NUM_VAL          = 1
         Mandatory        = "TRUE"
    END_OBJECT = QAPercentOutofBoundsData

    OBJECT = QAPercentMissingData
         Data_Location    = "PGE"
         TYPE             = "INTEGER"
         NUM_VAL          = 1
         Mandatory        = "FALSE"
    END_OBJECT = QAPercentMissingData

END_GROUP = QAStats

GROUP = SensorCharacteristic

    OBJECT = SensorCharacteristicContainer
        CLASS         = "M"     /* allows for many sensorcharacteristics */
                                /* per granule */
        Mandatory     = "TRUE"
        Data_Location = "NONE"

        OBJECT = SensorShortName
            Data_Location     = "PGE"
            CLASS             = "M"
            TYPE              = "STRING"
            NUM_VAL           = 1
            Mandatory         = "FALSE"
        END_OBJECT = SensorShortName

        OBJECT = InstrumentShortName
            Data_Location     = "PGE"
```

```
            CLASS             = "M"
            TYPE              = "STRING"
            NUM_VAL           = 1
            Mandatory         = "FALSE"
        END_OBJECT = InstrumentShortName

    END_OBJECT = SensorCharacteristicContainer

  END_GROUP = SensorCharacteristic

  GROUP = InformationContent

    OBJECT = InformationContentContainer
        CLASS         = "M"     /* counter for each additional attribute */
        Mandatory     = "TRUE"
        Data_Location = "NONE"

        OBJECT = ParameterName
            Data_Location     = "PGE"
            TYPE              = "STRING"
            CLASS             = "M"
            NUM_VAL           = 1
            Mandatory         = "FALSE"
        END_OBJECT = ParameterName

        OBJECT = ParameterValue
            Data_Location     = "PGE"
            CLASS             = "M"
            TYPE              = "DATETIME"
            NUM_VAL           =  1
            Mandatory         = "FALSE"
        END_OBJECT = ParameterValue

    END_OBJECT = InformationContentContainer

  END_GROUP = InformationContent

END_GROUP = INVENTORYMETADATA


GROUP = ARCHIVEDMETADATA
  GROUPTYPE = MASTERGROUP

  GROUP = Browse

    OBJECT =  BrowseSize
        Data_Location  = "PGE" /* may also be MCF (static) provided */
                               /* if not prone to change */
                               /* equal to MCF */
        TYPE           = "DOUBLE"
        NUM_VAL        = 1
        Mandatory      = "FALSE"
    END_OBJECT = BrowseSize
```

```
  END_GROUP = Browse
END_GROUP = ARCHIVEDMETADATA
END
```

Appendix D: Sample Inventory Metadata

```
GROUP                  = INVENTORYMETADATA
  GROUPTYPE            = MASTERGROUP

  OBJECT                 = SHORTNAME
    VALUE                = "COGGEOAA"
    NUM_VAL              = 1
  END_OBJECT             = SHORTNAME

  OBJECT                 = VERSIONID
    VALUE                = "1"
    NUM_VAL              = 1
  END_OBJECT             = VERSIONID

  OBJECT                 = SIZEMBECSDATAGRANULE
    NUM_VAL              = 1
    VALUE                = "NOT SET"
  END_OBJECT             = SIZEMBECSDATAGRANULE

  GROUP                  = INPUTGRANULE

    OBJECT                 = INPUTPOINTER
      NUM_VAL              = 1
      VALUE                = "NOT SET"
    END_OBJECT             = INPUTPOINTER

  END_GROUP              = INPUTGRANULE

  GROUP                  = RANGEDATETIME

    /* Format of RangeBeginningDate is YYYY-MM-DD or YYYY-DDD */

    OBJECT                 = RANGEBEGINNINGDATE
      NUM_VAL              = 1
      VALUE                = "1997-02-25"
    END_OBJECT             = RANGEBEGINNINGDATE


    /* Format of RangeBeginningTime is HH:MM:SS.SSSS... */

    OBJECT                 = RANGEBEGINNINGTIME
      NUM_VAL              = 1
      VALUE                = "10:16:57.0"
    END_OBJECT             = RANGEBEGINNINGTIME


    /* Format of RangeEndingDate is YYYY-MM-DD or YYYY-DDD */

    OBJECT                 = RANGEENDINGDATE
      NUM_VAL              = 1
      VALUE                = "NOT SET"
```

```
  END_OBJECT             = RANGEENDINGDATE


  /* Format of RangeEndingTime is HH:MM:SS.SSSS... */

  OBJECT                 = RANGEENDINGTIME
    NUM_VAL              = 1
    VALUE                = "NOT SET"
  END_OBJECT             = RANGEENDINGTIME

END_GROUP              = RANGEDATETIME

GROUP                  = ZONEIDENTIFIERCLASS

  OBJECT                 = ZONEIDENTIFIER
    NUM_VAL              = 6
    VALUE                = (10, 20, 30)
  END_OBJECT             = ZONEIDENTIFIER

END_GROUP              = ZONEIDENTIFIERCLASS

GROUP                  = BOUNDINGRECTANGLE

  OBJECT                 = EASTBOUNDINGCOORDINATE
    NUM_VAL              = 1
    VALUE                = 220.199997
  END_OBJECT             = EASTBOUNDINGCOORDINATE

END_GROUP              = BOUNDINGRECTANGLE

GROUP                  = QASTATS

  OBJECT                 = QAPERCENTOUTOFBOUNDSDATA
    NUM_VAL              = 1
    VALUE                = 30
  END_OBJECT             = QAPERCENTOUTOFBOUNDSDATA

END_GROUP              = QASTATS

GROUP                  = INFORMATIONCONTENT

  OBJECT                 = INFORMATIONCONTENTCONTAINER
    CLASS                = "1"

    OBJECT                 = PARAMETERNAME
      CLASS                = "1"
      NUM_VAL              = 1
      VALUE                = "ProductionDateTime"
    END_OBJECT             = PARAMETERNAME

    OBJECT                 = PARAMETERVALUE
      CLASS                = "1"
      NUM_VAL              = 1
      VALUE                = "1997-02-25T10:16:57.000000Z"
```

```
      END_OBJECT             = PARAMETERVALUE

    END_OBJECT               = INFORMATIONCONTENTCONTAINER

    OBJECT                   = INFORMATIONCONTENTCONTAINER
      CLASS                  = "2"

      OBJECT                   = PARAMETERNAME
        CLASS                  = "2"
        NUM_VAL                = 1
        VALUE                  = "PlatformShortName"
      END_OBJECT               = PARAMETERNAME

      OBJECT                   = PARAMETERVALUE
        CLASS                  = "2"
        NUM_VAL                = 1
        VALUE                  = "TRMM"
      END_OBJECT               = PARAMETERVALUE

    END_OBJECT               = INFORMATIONCONTENTCONTAINER

  END_GROUP                = INFORMATIONCONTENT

END_GROUP                = INVENTORYMETADATA

END
```