## 6.3 SDP Toolkit Tools—Optional

### 6.3.1 Digital Elevation Model Tools

#### 6.3.1.1 DEM Access Tools (HDF-based tools)

The Digital Elevation Model (DEM) access tools described in this section were introduced for the first time in TK 5.2 of the SDP Toolkit. They have been implemented in response to EOS Science Working Group - AM Platform (SWAMP) and ESDIS requests. These tools are meant to replace the access tools described in Section 6.3.1.2 for DEM access. The older access tools and associated ancillary data will continue to be distributed with the Toolkit, as long as there is an identifiable user requirement for them. Please note that the primary ECS production DEMs will supplied in HDF-EOS format and will be accessible through the tools in this section. The older ETOP05 data sets will be accessible in the production system through tools described in Section 6.3.2.

The DEM Toolkit tools in Section 6.3.1.1 are intended for accessing a hierarchy of DEM data sets. In order to utilize these functions, a user must install the SDP Toolkit on their machine. This hierarchy of data sets will include data from multiple resolutions. The DEM tools accesses this information based on resolution; a user indicates from which resolutions they are interested in query data. Each of these resolutions consists of multiple files. For example, the 3 arc second resolution data set (~100 m postings) is divided into 648 (10 degree by 10 degree) files. The number and extent of these files are transparent to the user. The user indicates interest in a particular resolution with a resolution tag. This resolution tag is initialized by the tool PGS_DEM_Open. The resolution tags MUST be initialized, either individually or as an array of the resolution tags, BEFORE any of the other DEM tools may access the data set at that resolution. These initialized resolution tags allow access of the underlying files (in the case of the 3 arc second resolution, the 10 degree by 10 degree files), without having to actually specify the particular physical file.

As mentioned above, the DEM tools may be used with a hierarchy of DEM data sets. Most of the DEM tools not only are able to accept a single resolution tag, but they may even accept a list, an array, of resolution tags. The first element of the array is the tag for the preferred resolution of the data (generally this will be the highest resolution data set). Each successive entry in the array will be in descending interest of use: in general, lower spatial resolution. If one inputs an array of resolution tags to a DEM tool, then one may be able to gain information across resolutions. For example, one may enter an array of resolution tags into the tool PGS_DEM_GetRegion. This tool will go to the data set files of the first resolution tag and extract the region of interest. If any of the points in the region of interest is a fill value, then the tool will access the next data set in the input array (for that particular point). It will continue to step through progressively lower resolution data sets (depending on the order of the elements in the inputted array) until it finds "valid", actual, non fill value, data.

The data sets supported by SDP Toolkit 5.2.14 are the 3 arc second (~100 m postings), 30 arc second (approximately 1km postings) and 90 arc second (approximately 3 km postings)

resolution data sets. The layer available in all resolutions is elevation (PGSd_DEM_ELEV). Other layers available in both 3 arc second and 30 arc second resolutions are water/land (PGSd_DEM_WATER_LAND), slope (PGSd_DEM_SLOPE), aspect (PGSd_DEM_ASPECT), standard deviation elevation (PGSd_DEM_STDEV_ELEV), and standard deviation slope (PGSd_DEM_STDEV_SLOPE). Also all resolutions include geoid (PGSd_DEM_GEOID). In addition, the 30 arc second data files include quality data such as data source (PGSd_DEM_SOURCE) and quality metric (PGSd_DEM_HORIZONTAL_ACCURACY) and PGSd_DEM_VERTICAL_ACCURACY). All data sets are in HDF-EOS GRID format. The 3 arc second resolution data set is divided into 648 (10 degree by 10 degree) tiles. For each tile there are 2 files, one that includes data for elevation, land/sea mask, slope, aspect, and geoid, and another file that includes data for the standard deviations. Only a few tiles are provided at the 3 arc second resolution, as test data. Full 3 arc data set resides at EDC DAAC. The 30 arc second resolution data set divides the Earth's surface into 6 tiles (2 files per tile as the 3 arc second data set). The 90 arc second resolution data covers the entire globe in one tile and includes all the ice shelfs for the Antartica that is in the latest Antartica version from the Radarsat Antartica Mapping Program (RAMP). These are delivered with the Toolkit as a provisional data set; updates are possible, for example to replace regions of fill value with actual data. All resolutions are in a Geographic Projection. By geographic, we mean that degrees of latitude and longitude are linearly mapped to row and column pixels, respectively.

To access these data sets, they must be included in the PCF. The files which make up the 30 arc second resolution should each have a logical ID equal to 10650 for the first file and 10651 for the second file. The logical ID of the 3 arc second resolution files should be 10653 for the first file and 10654 for the second file. The logical ID for the 90 arc second resolution file is 10656. For more information on setting up a PCF for DEM access, see both the DEM data set README file and the PCF template which accompanies Toolkit 5.2.14.

The DEM access tools described in Section 6.3.1.1 are:

**PGS_DEM_Open():** Open the DEM

**PGS_DEM_Close():** Close the DEM

**PGS_DEM_DataPresent():** Check for Valid DEM Data Point

**PGS_DEM_SortModels():** Check for Data in a Specified Region of the DEM

**PGS_DEM_GetPoint():** Return Data at Specified DEM Points

**PGS_DEM_GetRegion():** Return Data from a Specified Region of the DEM

**PGS_DEM_GetMetadata():** Extract Metadata from the DEM

**PGS_DEM_GetQualityData():** Access DEM Quality Data

**PGS_DEM_GetSize():** Return Size of Specified DEM Region

# Open the DEM

**NAME:** **PGS_DEM_Open()**

**SYNOPSIS:**

C:

```
#include <PGS_DEM.h>
PGSt_SMF_status
PGS_DEM_Open(
        PGSt_DEM_Tag      resolutionList[],
        PGSt_integer  numResolutions,
        PGSt_integer  layerList[],
        PGSt_integer  numLayers)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_open(resolutionList, numResolutions
                              layerList, numLayers)
        integer       resolutionList(*)
        integer       numResolutions
        integer       layerList(*)
        integer       numLayers
```

**DESCRIPTION:** This tool initializes a list of resolutions tags which correspond to a series of DEM data sets. These initialized resolution tags are used by the DEM tools. A DEM data set includes all the files of a particular resolution. Presently, only three data sets are available: 3 arc second, 30 arc second, and 90 arc second resolutions which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_30ARC, and PGSd_DEM_90ARC, respectively. A resolution tag MUST be initialized before it may be used in any of the other PGS_DEM tools. Each layer indicated in the layerList will automatically be initialized across all resolutions in the resolutionList. Note that for 90 arc second resolution the only available layer is elevation.

**INPUTS:**

resolutionList[] -- an array of resolution tags, data sets. See Notes.

numResolutions -- the number of resolution tags in the array resolutionList

layerList[] -- the DEM layers to initialize. See Notes.

numLayers -- the number of DEM Layers in the list.

**OUTPUTS:** N/A

**RETURNS:** PGS_S_SUCCESS -- success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_integer numLayers;

PGSt_integer layerList[2];

PGSt_integer numResolutions;

PGSt_DEM_Tag resolutionList[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;

resolutionList[1]= PGSd_DEM_30ARC;

numResolutions = 2;

layerList[0] = PGSd_DEM_ELEV;

layerList[1] = PGSd_DEM_WATER_LAND;

numLayers = 2;

/* Open the resolutions and data layer*/

status = PGS_DEM_Open(resolutionList, numResolutions,
layerList, numLayers);

if (status != PGS_S_SUCCESS)

{

/* Do some error handling ... */
```

FORTRAN:

```
                integer     numLayers

                integer     numResolutions

                integer     layerList(2)

                integer     resolutionList(2)

                integer     status

C INITIALIZE

                resolutionList(1) = PGSd_DEM_3ARC

                resolutionList(2) = PGSd_DEM_30ARC

                layerList(1) = PGSd_DEM_ELEV

                layerList(2) = PGSd_DEM_WATER_LAND

                numResolutions = 2

                numLayers = 2

                status = pgs_dem_open(resolutionList, numResolutions,

     1                 layerList, numLayers)
```

**NOTES:** resolutionList:

For earlier ECS Deliveries and SCFToolkits 5.2.2-5.2.7, the data sets that may be inputted are 3 arc second, 30 arc second sets which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_30ARC, respectively.

layerList:

For ECS Deliveries Drop 4 and later, the only layer that may be inputted for the 3 arc and 30 arc second resolution are elevation, (PGSd_DEM_ELEV), water/land (PGSd_DEM_WATER_LAND), standard deviation elevation (PGSd_DEM_STDEV_ELEV), slope (PGSd_DEM_SLOPE), standard deviation slope (PGSd_DEM_STDEV_SLOPE), and aspect (PGSd_DEM_ASPECT). The other layers that will be available are topographical obscuration (PGSd_DEM_TOP_OBSC). and topographical shadow (PGSd_TOP_SHAD). For 90 arc second resolution the only available layer is elevation.

**REQUIREMENTS:** PGSTK–0940

# Close the DEM

**NAME:** **PGS_DEM_Close()**

**SYNOPSIS:**

C:

```
#include <PGS_DEM.h>
PGSt_SMF_status
PGS_DEM_Close(
        PGSt_DEM_Tag      resolutionList[],
        PGSt_integer  numResolutions,
        PGSt_integer  layerList[],
        PGSt_integer  numLayers)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_close(resolutionList, numResolutions
1 layerList, numLayers)
integer      resolutionList(*)
integer      numResolutions
integer      layerList(*)
integer      numLayers
```

**DESCRIPTION:** This tool closes the session begun by the tool PGS_DEM_Open. One can close multiple data set sessions simultaneously or independently. If one wants to only close one DEM data set, the array resolutionList should only contain an individual resolution tag. Presently, only three data sets are available: 3 arc second (small test data set), 30 arc second (provisional global data), and 90 arc second resolutions which correspond to the tags PGSd_DEM_3ARC, PGSd_DEM_30ARC, and PGS_DEM_90ARC respectively. Each layer in the layerList will automatically be closed across all the resolutions indicated in the resolutionList.

**INPUTS:** resolutionList[] - an array of resolution tags, data sets. See Notes to PGS_DEM_Open().

numResolutions - the number of resolution tags in the array resolutionList.

layerList[] - the number of DEM Layers to initialize. See Notes to PGS_DEM_Open().

numLayers - the number of DEM Layers in the list.

**OUTPUTS:** N/A

**RETURNS:** PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_integer numLayers;

PGSt_integer layerList[2];

PGSt_integer numResolutions;

PGSt_DEM_Tag resolutionList[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;

resolutionList[1]= PGSd_DEM_30ARC;

numResolutions = 2;

layerList[0] = PGSd_DEM_ELEV;

numLayers = 1;

/* Close the resolutions and data layer*/

status = PGS_DEM_Close(resolutionList, numResolutions,
layerList, numLayers);
```

FORTRAN:

```
                  integer     numLayers
                  integer     numResolutions
                  integer     layerList(2)
                  integer     resolutionList(2)
                  integer     status
C INITIALIZE
                  resolutionList(1) = PGSd_DEM_3ARC
                  resolutionList(2) = PGSd_DEM_30ARC
                  layerList(1) = PGSd_DEM_ELEV
                  numResolutions = 2
                  numLayers = 1
                  status = pgs_dem_close(resolutionList, numResolutions,
     1                  layerList, numLayers)
```

**REQUIREMENTS:** PGSTK–0948

# Check for Valid DEM Data Point

**NAME:** **PGS_DEM_DataPresent()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_DataPresent(
PGSt_DEM_Tag resolution,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double pntLatitude[],
PGSt_double pntLongitude[],
PGSt_integer numPoints,
PGSt_boolean *dataPresent)
```

FORTRAN:

```
      include <PGS_SMF.f>
      include <PGS_DEM.f>
      include <PGS_DEM_14.f>
      include <PGS_MEM_7.f>

      integer function pgs_dem_datapresent(resolution, layer,
     1 positionCode, pntLatitude, pntLongitude, numPoints, dataPresent)
      integer resolution
      integer        layer
      integer positionCode
      double precision      pntLatitude(*)
      double precision      pntLongitude(*)
      integer numPoints
      integer dataPresent
```

**DESCRIPTION:** This tool checks whether pixel(s), at specified latitude(s) and longitude(s), are data or fill values. In dataPresent, either PGS_TRUE or

PGS_FALSE will be returned, corresponding to valid data or fill value, respectively.

**INPUTS:** resolution - the resolution tag for a particular data set. An element of the array resolutionList which is initialized by PGS_DEM_Open. See Notes to PGS_DEM_Open().

layer - indicates which data mask or layers one is accessing. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes.

pntLatitude[] and pntLongitude[] - the latitude and longitude of the point(s) of interest. See Notes.

numPoints - the number of points to be queried.

**OUTPUTS:** dataPresent - either PGS_TRUE or PGS_FALSE. PGS_TRUE indicates that a "valid" data value is at the specific location(s). PGS_FALSE indicates that there is a fill value in the extracted data.

**RETURNS:** PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_SMF_status status;
PGSt_integer layer;
PGSt_DEM_Tag resolution;
PGSt_integer numDataPoints;
PGSt_boolean dataPresent;
PGSt_double pntLatitude[3];
PGSt_double pntLongitude[3];

/* initialize input parameters, both resolutions and layer */

resolution= PGSd_DEM_3ARC;

layer = PGSd_DEM_ELEV;


/* initialize the number of data points and pick position of points one
interested in.  In this case, positions are in signed
decimal degrees  */

numDataPoints = 3;
```

```
                pntLatitude[0] =   40.05;

                pntLongitude[0] = -105.3

/*see if selected points have real data*/

                status     =     PGS_DEM_DataPresent(resolution,     layer,
                PGSd_DEM_DEGREE,  pntLatitude,  pntLongitude,  numDataPoints,
                &dataPresent);
```

FORTRAN:

```
                integer     layer

                integer     resolution

                integer status

                integer numDataPoints

                integer dataPresent

                double precision pntLatitude(3)

                double precision pntLongitude(3)

C INITIALIZE resolution and layers

                resolution = PGSd_DEM_3ARC

                layer = PGSd_DEM_ELEV


C INITIALIZE points of interest. in this case, in signed decimal degrees

                numDataPoints = 3

                pntLatitude(1) = 40.04

                pntLongitude(1) = -105.3

                status = pgs_dem_datapresent(resolution, layer,

     1          PGSd_DEM_DEGREE, pntLatitude, pntLongitude,

     1          numDataPoints, dataPresent)
```

**NOTES:** layer:

See NOTES for layerList of PGS_DEM_OPEN.

positionCode:

The position inputs may be either in signed, decimal degree format or in global pixel format, which correspond to the flags PGSd_DEM_DEGREE and PGSd_DEM_PIXEL, respectively. NOTE: global pixel format is the pixel coordinates of a point in the coordinate system for the whole world. This is NOT the same as pixels inside a single HDF-EOS GRID. The pixel coordinate system is unique for each resolution. The origin of all the pixel coordinate systems is the North, West corner of the globe (180W, 90N). The coordinate system is zero based. The 30 arc second resolution has 43200 pixels spanning from 180 West to 180 East and 21600 pixels spanning from North Pole to South Pole. The 3 arc second resolution has 432000 pixels spanning from 180 West to 180 East and 216000 pixels spanning from North Pole to South Pole. The 90 arc second resolution has 14400 pixels spanning from 180 West to 180 East and 7200 pixels spanning from North Pole to South Pole.

pntLatitude and pntLongitude:

Each longitude point MUST have a corresponding latitude point. The latitude(s) and longitude(s) will be in either signed, decimal degree format or global pixel format, corresponding to the flag indicated by positionCode. If the user is using the flag PGSd_DEM_PIXEL, they should be aware that the values for pntLatitude and pntLongitude will be truncated. In other words, if the user passed in a pixel position which had any decimal information, that information would NOT be used in accessing the data. For example, if the user passed in 1267.34 as a pixel position, it would be truncated to 1267.

**REQUIREMENTS:** PGSTK–0941

# Check for Data in a Specified Region of the DEM

**NAME:** **PGS_DEM_SortModels()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_SortModels(
PGSt_DEM_Tag resolutionList[],
PGSt_integer  numResolutions,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
PGSt_DEM_Tag *completeDataSet)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_sortmodels(resolutionList,numResolutions,
1  layer, positionCode, latitude, longitude,   completeDataSet)
integer          resolutionList(*)
integer          numResolutions
integer          layer
integer          positionCode
double precision        latitude(2)
double precision        longitude(2)
integer          completeDataSet
```

**DESCRIPTION**: This tool will check the DEM data sets for complete data in a rectangular region defined by the latitude/longitude pair specified (i.e., upper left hand

corner, lower right hand corner). If there are fill values at any of the points in the defined region, then the tool will query the next resolution tag in the array for that region. The first DEM data set to have complete data in the region of interest will have its corresponding resolution tag returned in completeDataSet. If none of the data sets in the input array is "complete", then the PGSd_DEM_NO_COMPLETE_DATA will be returned.

**INPUTS:** resolutionList[] - an array of resolution tags, data sets. See Notes to PGS_DEM_Open().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, latitude and longitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the bounding latitudes and longitudes of the region of interest. See Notes.

**OUTPUTS:** completeDataSet - pointer to a resolution tag, data set identifier. The first DEM data set to have complete data in the region of interest will be returned. If none of the resolution tags in the inputted array is "complete", then the PGSd_DEM_NO_COMPLETE_DATA will be returned.

**RETURNS:** PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList;

PGSt_integer numResolutions;

PGSt_integer completeData;

PGSt_double latitude[2];

PGSt_double longitude[2];

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;
```

```
                  resolutionList[1] = PGSd_DEM_30ARC;

                  layer = PGSd_DEM_ELEV;

/* initialize the upper left and lower right corners of the data region..  In
                  this case, positions are in signed decimal degrees */

/*upper left corner*/

                  latitude[0] =  44.0;

                  longitude[0] = -80.0;

/*lower right corner*/

                  latitude[1] =  43.0;

                  longitude[1] = -79.0;

/* see if region has real data */

                  status = PGS_DEM_SortModels(resolutionList, numResolutions,
                  layer, PGSd_DEM_DEGREE, latitude, longitude, &completeData);

                  if (status!= PGS_S_SUCCESS)

/* Do some error handling ...*/

                  else


/*  compare complete data set to the three possibilities to find resolution
                  which has complete data across this region.

*/

                  if (completeData == PGSd_DEM_3ARC)


/* complete region found in 3 arc second resolution */

                  }

                  else if (completeData == PGSd_DEM_30ARC)

                  {

/* complete region in 30 arc second resolution */

...

                  }

                  else if (completeData == PGSd_DEM_NO_COMPLETE_DATA)
```

```
                  {
/* all resolutions contained fill values within this region */
...
                  }
```

FORTRAN:

```
                  integer status
                  integer layer
                  integer resolutionList
                  double precision latitude(2)
                  double precision longitude(2)
C  initialize input parameters, both resolutions and layer
                  resolutionList(1)= PGSd_DEM_3ARC
                  resolutionList(2) = PGSd_DEM_30ARC
                  layer = PGSd_DEM_ELEV

C  initialize the upper left and lower right corners of the data
C region..  In this case, positions are in signed decimal degrees

C upper left corner
                  latitude(1) =  44.0
                  longitude(1) = -80.0
C  lower right corner
                  latitude(2) =  43.0
                  longitude(2) = -79.0

C  see if region has complete data
                  status = pgs_dem_sortmodels(resolutionList,
     1            numResolutions, layer, PGSd_DEM_DEGREE, latitude, longitude,
                  completeData)
                  if (status.NE.PGS_S_SUCCESS) then
```

```
C Do some error handling
                  else

C compare completeData to determine the resolution with complete data
C  in the specified region

                  if (completeData.EQ.PGSd_DEM_3ARC) then

C                       complete data in 3 arc second resolution
....
                  elseif (completeData.EQ.PGSd_DEM_30ARC) then

C                       complete data found in 30 arc second resolution
...
                  elseif (completeData.EQ.PGSd_DEM_NO_COMPLETE_DATA) then

C                 all resolutions contained fill values within this
C                  region
```

**NOTES:** latitude and longitude:

The first point corresponds to the upper left corner of the rectangular region, and the second point correspond to the lower right corner of the rectangular region. The latitude(s) and longitude(s) will be in either signed, decimal degree format or global pixel format, corresponding to the flag indicated by positionCode. If the user is using the flag PGSd_DEM_PIXEL, she or he should be aware that the values for latitude and longitude will be truncated. In other words, if the user passed in a pixel position which had any decimal information, that information would NOT be used in accessing the data. For example, if the user passed in 1267.34 as a pixel position, it would be truncated to 1267.

**REQUIREMENTS:** PGSTK–0942

# Return Data at Specified DEM Points

**NAME:** **PGS_DEM_GetPoint()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_GetPoint(
PGSt_DEM_Tag  resolutionList[],
PGSt_integer  numResolutions
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_double pntLatitude[],
PGSt_double pntLongitude[],
PGSt_integer numPoints
PGSt_integer interpolation,
void *interpValues)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getpoint(resolutionList, numResolutions, layer,
positionCode, pntLatitude, pntLongitude, numPoints, interpolation,
interpValue)
integer          resolutionList(*)
integer          numResolutions
integer          layer
integer positionCode
double precision        pntLatitude(*)
double precision        pntLongitude(*)
```

integer numPoints

integer interpolation

‘user defined’ interpValue(*)

**DESCRIPTION:** This tool attempts to return the data value(s) of the point(s) defined by latitude and longitude. If the latitude and longitude do not exactly correspond to the center (or corner, depending on the manner in which the DEM map has been constructed) of a pixel, the value will be interpolated. Presently, there are only two interpolation methods supported: nearest neighbor and bilinear interpolation. If at this point there is a "hole", a fill value, in the data set, then the tool will access the next resolution tag in the input array. It will continue to step through progressively lower resolution data sets (depending on the order of the elements in the inputted array) until it finds actual data for that point. If all of the DEM data sets have a "hole" at this particular location, then the PGSDEM_M_FILLVALUE_INCLUDED will be returned. Even if some of the queried points are not able to be interpolated (i.e. at the lowest resolution that region is fill value), the value, interpolated value or fill value of the point(s) will be returned in interpValues.

**INPUTS:** resolutionList - an array of resolution tags, data sets. See Notes to PGS_DEM_SortModels().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

pntLatitude[] and pntLongitude[] - the latitude and longitude of the point of interest. See Notes to PGS_DEM_DataPresent().

numPoints - the number of points to be queried.

interpolation - type of interpolation. see Notes.

**OUTPUTS:** interpValues - the data value(s) at the designated latitude(s) and longitude(s). See Notes.

**RETURNS**: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

PGSDEM_M_FILLVALUE_INCLUDED - fill values in the returned data

PGSDEM_M_MULTIPLE_RESOLUTIONS - data accessed from multiple resolutions

**EXAMPLES:**

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList[2];

PGSt_integer numResolutions;

PGSt_integer numDataPoints;

PGSt_double pntLatitude[10];

PGSt_double pntLongitude[10];

short dataPoints[10)];

/* NOTE: The type of data buffer should correspond to the type of data that
one is extracting.  Presently, the only available data are
PGSd_DEM_ELEV,  PGSd_DEM_SLOPE,  PGSd_DEM_ASPECT,
PGSd_DEM_STDEV_ELEV,  PGSd_DEM_STDEV_SLOPE,  and
PGSd_DEM_WATER_LAND which are of type 2 byte, 1 byte, 2
byte, 2 byte, 2 byte, and 2 byte integers, respectively.  In
the future, there will be data layers added which are NOT 2
byte or 1 byte integers.  If one does not know the data type
of the particular layer, then one should use the tool
PGS_DEM_GetSize.*/


/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;

resolutionList[1] = PGSd_DEM_30ARC;

layer = PGSd_DEM_ELEV;


/* initialize the location of the points of interest.  In this case, positions
are in signed decimal degrees*/

pntLatitude[0] =  40.05;

pntLongitude[0] = -105.3;...
```

```
status = PGS_DEM_GetPoint(resolutionList, numResolutions,
layer, PGSd_DEM_DEGREE, pntLatitude, pntLongitude,
numDataPoints, PGSd_DEM_NEAREST_NEIGHBOR, (void
*)dataPoints);


/*Possible status returns*/

if (status == PGS_S_SUCCESS)

{

/*no fill points*/

...

}

else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

{

/*fill points included in the extracted data*/

...

}

else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)

{

/*no fill points in data buffer, fill points interpolated from multiple
resolutions*/

...

}

else

{

/*Error in extracting the data */

/* Do some error handling*/
```

FORTRAN:

```
integer status

integer layer
```

```
                  integer resolutionList(2)
                  integer numResolutions
                  integer numDataPoints
                  double precision pntLatitude(10)
                  double precision pntLongitude(10)
                  integer*2 dataPoints(10)
C *** NOTE: The type of data buffer should correspond to the type of
C data one is extracting.  Presently, the only available data are
C PGSd_DEM_ELEV, PGSd_DEM_WATER_LAND, PGSd_DEM_SLOPE, PGSd_DEM_ASPECT,
C PGSd_DEM_STD_DEV_ELEV, and PGS_DEM_STDEV_SLOPE which are of type
C 2 byte integers (except for PGSd_DEM_WATER_LAND which is 1 byte
C integer).
C In the future, there will be data layers added which are NOT 2 byte
C or 1 byte integers.  If one does not know the data
C type of the particular
C layer, then one should use the tool PGS_DEM_GetSize.

C initialize input parameters, both resolutions and layer

                  resolutionList(1)= PGSd_DEM_3ARC
                  resolutionList(2) = PGSd_DEM_30ARC
                  layer = PGSd_DEM_ELEV

C initialize points of intereset.  In this case, location is in signed
C decimal  degrees.
                  PntLatitude(0) =  40.05
                  pntLongitude(0) = -105.3
                  status = pgs_dem_getpoint(resolutionList,
     1            numResolutions, layer, PGSd_DEM_DEGREE, pntLatitude,
```

```
     1            pntLongitude, numDataPoints,
     1            PGSd_DEM_NEAREST_NEIGHBOR, dataPoints)

C  Possible status returns

                  if (status .EQ. PGS_S_SUCCESS) then

C  no fill values in this region, in the first resolution
                  ...
                  elseif (status .EQ. PGSDEM_M_FILLVALUE_INCLUDED) then
C  fill values included in extracted data
                  ...
                  elseif (status .EQ. PGSDEM_M_MULTIPLE_RESOLUTIONS) then
C  no fill values included in extracted data.  All fill values
C interpolated  from other resolutions in resolutionList

                  else

C  Error extracting data
C  Do some error handling ...
```

**NOTES:** All the 30 arc second, 3 arc second, and 90 arc second DEM data are referenced vertically to mean sea level, which is approximated by the geoid. Thus, the elevation data retrieved by PGS_DEM_GetPoint tool will be with respect to the mean sea level. To get height relative to the WGS84 ellipsoid see note for the function PGS_DEM_GetQualityData.

interpolation:

Presently there is only one type of interpolation, nearest neighbor, PGSd_DEM_NEAREST_NEIGHBOR, and bilinear interpolation, PGSd_DEM_BILINEAR.

interpValues:

If the function locates fill values in the extracted data from the first resolution in the resolutionList, it will attempt to interpolate from the other resolutions. If the point of interest corresponds to a fill value at the lowest resolution (the last resolution tag of resolutionList), then this fill value(s) will be returned.

The land/water classes are described below:

0. Shallow Ocean (Ocean <5k from coast OR <50m deep; i.e., a buffer zone around all coastal areas and islands, plus shallow areas up to 50m deep that are further than 5km from the land). Includes the appropriate parts of the Black Sea, Red Sea, Mediterranean Sea, Hudson Bay, and other ocean-connected seas.
1. Land (not anything else).
2. Ocean Coastlines and Lake Shorelines (an actual boundary line).
3. Shallow Inland Water (Inland Water <5km from shore OR <50m deep; i.e., a buffer zone around all lake shores and inland islands, plus shallow areas up to 50m deep that are further than 5km from the land). Includes the appropriate parts of the Caspian Sea, Aral Sea, Great Lakes, "2-line" rivers, etc.
4. Ephemeral (intermittent) Water (from Digital Chart of the World).
5. Deep Inland Water (Inland water >5km from shoreline AND >50m deep; i.e., Lake waters beyond 5km from their shore or islands, and greater than 50m eep). Includes the appropriate parts of the Caspian Sea, Aral Sea, Great akes, etc.
6. Continental Shelf Ocean (Ocean >5km from coast AND between 50m and 500m deep); i.e., Oceans beyond 5km from coastal areas and islands, and greater than 50m deep but less than 500m deep. Primarily represents the Continental shelf areas.
7. Deep Ocean (Ocean >5km from coast AND >500m deep); i.e., The really deep oceans.

IMPORTANT!! It is the user's responsibility to allocate the appropriate amount of space for interpValue. Note, that each mask has its own data type, see PGS_DEM_GetSize.

WARNING: Because of memory limitations it is not possible to extract more than a certain number of points by a single call to this function. The maximum number of points that can be extracted by one call to this function depends on the machine configuration at the runtime.

**REQUIREMENTS:** PGSTK–0943

## Return Data from a Specified Region of the DEM

**NAME:** **PGS_DEM_GetRegion()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_GetRegion(
PGSt_DEM_Tag  resolutionList[],
PGSt_integer numResolutions,
PGSt_integer layer,
PGSt_integer positionCode,
PGSt_integer interpolation,
PGSt_double latitude[2],
PGSt_double longitude[2],
void *dataRegion,
PGSt_double regionSize[2],
PGSt_double firstElement[2],
PGSt_double pixelSize[2])
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getregion(resolutionList, numResolutions,
layer, positionCode, interpolation, latitude, longitude, dataRegion,
regionSize, firstElement, pixelSize)
integer        resolutionList(*)
integer        numResolutions
integer        layer
integer        positionCode
```

| | |
|---|---|
| integer | interpolation |
| double precision | latitude(2) |
| double precision | longitude(2) |
| ‘user defined’ | dataRegion(*) |
| double precision | regionSize(2) |
| double precision | firstElement(2) |
| double precision | pixelSize(2) |

**DESCRIPTION:** This tool returns the data from a rectangular region of the DEM data set. In addition to returning an array of data, this tool will return the dimension of the region in terms of coordinate degrees, the coordinates of the first element of the dataRegion, and the size of the pixel. If any of the points in the region of interest is a "hole", a fill value, then the tool will access the next DEM data set in the input array. It will continue to step through progressively lower resolution data sets (depending on the order of the resolution tags in the inputted array) until it finds "valid", actual data. If all of the inputted resolutions have a "hole" at these specific locations, then the PGSDEM_M_FILLVALUE_INCLUDED will be returned. Even if some of the queried points are not able to be interpolated (i.e., at the lowest resolution that region is fill value), the data region is still returned. The only consequence is that dataRegion will not consist solely of "valid" and interpolated data but will also contain fill values

**INPUTS:** resolutionList - an array of resolution tags, data sets. See Notes to PGS_DEM_SortModels().

numResolutions - the number of resolution tags in the array resolutionList

layer - indicates which data mask or layer one is accessing. See Notes to PGS_DEM_DataPresent().

positionCode - flag indicating the format of the position inputs, latitude and longitude. See Notes to PGS_DEM_DataPresent().

. latitude[2] and longitude [2] - the bounding latitudes and longitudes of the region of interest. See Notes to PGS_DEM_SortModels().

. interpolation - type of interpolation. See Notes to PGS_DEM_GetPoint().

.**OUTPUTS:** dataRegion - an array in which the DEM data will be returned. See Notes.

regionSize[2] - an array indicating the size of the region in terms of the degrees of latitude and longitude. The array elements correspond to latitude and longitude respectively. The values will be in decimal format.

firstElement[2] - an array indicating the latitude and longitude, in decimal degree format, of the first element. The elements of the array correspond to latitude and longitude respectively.

pixelSize[2] - an array indicating the size of a pixel in terms of the degrees of latitude and longitude. The array elements correspond to latitude and longitude respectively.

**RETURNS**: PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

PGSDEM_M_FILLVALUE_INCLUDED - fill values in the returned data

PGSDEM_M_MULTIPLE_RESOLUTIONS - data accessed from multiple resolutions

**EXAMPLES:**

C:

```
PGSt_SMF_status status;

PGSt_integer layer;

PGSt_DEM_Tag resolutionList[2];

PGSt_integer numResolutions;

PGSt_double latitude[2];

PGSt_double longitude[2];

PGSt_double regionSize[2];

PGSt_double firstElement[2];

PGSt_double pixelSize[2];

short * dataRegion;

/* NOTE: The type of data buffer should correspond to the type of data one is
         extracting.  Presently, the only available data are
         PGSd_DEM_ELEV,  PGSd_DEM_WATER_LAND,  PGSd_DEM_SLOPE,
         PGSd_DEM_ASPECT,  PGSd_DEM_STD_DEV_ELEV,  and
         PGS_DEM_STDEV_SLOPE which are of type 2 byte integers
         (except for PGSd_DEM_WATER_LAND which is 1 byte integer).

/* initialize input parameters, both resolutions and layer */

resolutionList[0]= PGSd_DEM_3ARC;
```

```
                resolutionList[1] = PGSd_DEM_30ARC;

                layer = PGSd_DEM_ELEV;


/* initialize the location of the region of interest.  In this case, positions
                are in signed decimal degrees*/


/* upper left corner of region */

                pntLatitude[0] =  44.05;

                pntLongitude[0] =  -80.0;

/* lower right corner of region */

                latitude[1] = 43.0;

                longitude[1] = -78.8;


/*Allocate space for the buffers GetRegion.  It is the USER's RESPONSIBILITY
                TO ALLOCATE SPACE. .  If one does not know the data type or
                the extent of one's region in global pixels, then one should
                use the tool PGS_DEM_GetSize. */

                status  =  PGS_DEM_GetRegion(resolutionList,  numResolutions,
                layer ,PGSd_DEM_DEGREE, PGSd_DEM_NEAREST_NEIGHBOR, latitude,
                longitude, dataRegion, regionSize, firstElement, pixelSize);


/* possible status returns */

                if (status == PGS_S_SUCCESS)

                {

                /*no fill points*/

                ...

                }

                else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

                {

/*fill points included in the extracted data*/

                ...
```

```
                }
                else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)
                {
/*no fill points in data buffer, fill points interpolated from multiple
                resolutions*/
                ...
                }
                else
                {
/*Error in extracting the data */
/* Do some error handling*/
```

FORTRAN:

```
                integer status
                integer layer
                integer resolutionList(2)
                integer numResolutions
                double precision latitude(2)
                double precision longitude(2)
                double precision regionSize(2)
                double precision firstElement(2)
                double precision pixelSize(2)
                integer*2 dataRegion(*)
C *** NOTE: The type of data buffer should correspond to the type of data one
                is extracting.
C Presently, the only available data are PGSd_DEM_ELEV, PGSd_DEM_WATER_LAND,
C PGSd_DEM_SLOPE, PGSd_DEM_ASPECT, PGSd_DEM_STD_DEV_ELEV,
C and PGS_DEM_STDEV_SLOPE which are of type 2 byte integers (except
C for PGSd_DEM_WATER_LAND which is 1 byte integer).
```

```
C In C the future, there will be data layers added which are NOT 2 byte or 1
byte integers. It is

C the USER's RESPONSIBILITY TO ALLOCATE SPACE.  If one does not know the data
type C or C the extent of one's region in global pixels, then one should use
the tool C PGS_DEM_GetSize. ***


C   initialize input parameters, both resolutions and layer

                 resolutionList(1)= PGSd_DEM_3ARC

                 resolutionList(2) = PGSd_DEM_30ARC

                       layer = PGSd_DEM_ELEV


C  initialize the region of interest. In this case, the position is in signed
                 decimal degrees.

C  upper left corner of region

                 pntLatitude(1) =  44.05

                 pntLongitude(1) =  -80.0

C   lower right corner of region

                 latitude(2) = 43.0

                 longitude(2) = -78.8

                 status = PGS_DEM_GetRegion(resolutionList,

     1           numResolutions, layer ,PGSd_DEM_DEGREE,

     1           PGSd_DEM_NEAREST_NEIGHBOR, latitude, longitude,

     1            dataRegion, regionSize, firstElement, pixelSize)


C  possible status returns


                 if (status == PGS_S_SUCCESS)

C  **no fill points

                 else if (status == PGSDEM_M_FILLVALUE_INCLUDED)

C **fill points included in extracted data
```

```
                  else if (status == PGSDEM_M_MULTIPLE_RESOLUTIONS)

C  **no fill points in extracted data.  All fill points

C interpolated from other resolutions in resolutionList **

                  else

C  **Error extracting data

C  **Do some error handling ...
```

**NOTES:** dataRegion:

If the function locates fill values in the extracted data from the first resolution in the resolutionList, it will attempt to interpolate from the other resolutions. If the point of interest corresponds to a fill value at the lowest resolution (the last resolution tag of resolutionList), then this fill value(s) will be returned.

IMPORTANT!! It is the user's responsibility to allocate the appropriate amount of space for dataRegion. Note, that each mask has its own data type, see PGS_DEM_GetSize.

**REQUIREMENTS:** PGSTK–0944

# Extract Metadata from the DEM

**NAME:** **PGS_DEM_GetMetadata()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_GetMetadata(
PGSt_DEM_Tag resolution.
PGSt_integer layer,
PGSt_double pixLatInfo[2],
PGSt_double pixLonInfo[2],
char *positionUnits,
PGSt_double *scaling,
PGSt_double *offset,
PGSt_double *fillValue,
char *dataUnits,
PGSt_integer *mapProjection,
PGSt_boolean *qualityAssurLayer)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getmetadata(resolution, layer, pixLatInfo,
pixLonInfo, positionUnits, scaling, offset, fillValue, dataUnits,
mapProjection, qualityAssurLayer)
integer          resolution
integer          layer
double precision      pixLatInfo(2)
double precision      pixLonInfo(2)
character        positionUnits(*)
```

double precision scaling

double precision offset

double precision fillValue

character dataUnits(*)

integer mapProjection

integer qualityAssurLayer

**DESCRIPTION:** This tool accesses the general metadata that pertains to a single DEM data set The metadata is for the whole data set, not for isolated geographic sections of the data. Some of the metadata are valid for all the attributes, but other metadata will be mask specific.

**INPUTS:** resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

layer - indicates which data mask or layer one is accessing. See Notes to PGS_DEM_DataPresent().

**OUTPUTS:**

pixLatInfo - an array of information on the global row pixels. See Notes.

pixLonInfo - an array of information on the global column pixels. See Notes.

positionUnits - units of the position coordinates

scaling - a pointer to the scaling factor to convert attribute data to its appropriate units

offset - a pointer to an offset to convert the attribute data (after scaling) to a meaningful value

resolution - a pointer to the resolution of the attribute data

dataUnits - the units of the attribute data

fillValue - a pointer to the fill value of the specified attribute data

mapProjection - a pointer to the type of geographic projection applied to the attribute data. Corresponds to different projection flags. See HDF-EOS User's Guide for projection codes.

qualityAssurLayer - flag indicating a quality assurance and source layer for the attribute data. This will either have the value PGS_TRUE or PGS_FALSE which corresponds to the existence and the absence, respectively, of a quality assurance layer.

**RETURNS:** PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
                PGSt_SMF_status status;

                PGSt_integer layer;

                PGSt_DEM_Tag resolution;

                PGSt_double pixLatInfo[2];

                PGSt_double pixLonInfo[2];

                PGSt_double scaling;

                PGSt_double offset;

                PGSt_double fillValue;

                character *positionUnits;

                character *dataUnits;

                PGSt_integer mapProjection;

                PGSt_boolean qualityAssuranceLayer;

/* initialize resolution and layer*/

                resolution = PGSd_DEM_3ARC;

                layer = PGSd_DEM_ELEV;

/* allocate enough space for positionUnits and dataUnits string *

                positionUnits = calloc(30. sizeof(char));

                dataUnits = calloc(30. sizeof(char));

                status = PGS_DEM_GetMetadata(resolution, layer, pixLatInfo,
                pixLonInfo, positionUnits, &scaling, &offset, &fillValue,
                dataUnits, &mapProjection, &qualityAssuranceLayer);

                if (status != PGS_S_SUCCESS)

                {

/* Do some error handling  */
```

FORTRAN:

```
                integer status
                integer layer
                integer resolution
                double precision pixLatInfo(2)
                double precision pixLonInfo(2)
                double precision scaling
                double precision offset
                double precision fillValue
                integer mapProjection
                integer qualityAssuranceLayer
                character positionUnits(30)
                character dataUnits(30)
c   **Note: character arrays should have enough space allocated to hold the
                string.  THIS IS THE USER's RESPONSIBILITY ***
c  initialize resolution and layer
                resolution = PGSd_DEM_3ARC
                layer = PGSd_DEM_ELEV
                status = PGS_DEM_GetMetadata(resolution, layer,
     1          pixLatInfo, pixLonInfo, positionUnits, &scaling,
     1          &offset, &fillValue, dataUnits, &mapProjection,
     1          &qualityAssuranceLayer)
                if (status .NE. PGS_S_SUCCESS) then
c  ** Do some error handling
```

**NOTES:** pixLatInfo and pixLonInfo:

All of the values of this array are in degree decimal format. The first element of the array indicates the spacing between pixels. The second element is the location within the pixel that is used for requesting the location of that pixel (i.e. the center or corner of the pixel). This second element is the vertical (pixLatInfo) or horizontal (pixLonInfo) offset of this location from the top left corner of a pixel.

**REQUIREMENTS:** PGSTK–0945

# ACCESS DEM Quality Data

**NAME:** **PGS_DEM_GetQualityData()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_GetQualityData(
PGSt_DEM_Tag resolution,
PGSt_integer  qualityField,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
void *qualityData)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getqualitydata(resolution, qualityField,
positionCode, latitude, longitude, qualityData)
integer        resolution
integer        qualityField
integer positionCode
double precision      latitude(2)
double precision      longitude(2)
'user defined'  qualityRegion(*)
```

**DESCRIPTION:** This tool accesses the quality assurance layer of a particular DEM data set. It takes a latitude and longitude of a point of interest and an attribute mask. It returns information concerning the data source, the region over which the quality assurance information is valid, the quality metric of the aforesaid region, or information on the geoid.

**INPUTS:** resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

qualityField - the type of quality information requested. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the latitude and longitude of the points of interest in decimal format. See Notes to PGS_DEM_SortModels().

**OUTPUTS:** qualityData - an array containing the quality assurance layer information for the region specified. The information returned is dependent on the flag indicated in the qualityField. For example, one can obtain the data sources for all the data in one's region.

**RETURNS:** PGS_S_SUCCESS -- success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_SMF_status  status;
PGSt_integer numLayers;
PGSt_integer layerList[1];
PGSt_DEM_Tag resolutionList[1];
PGSt_integer numResolutions;
PGSt_double latitude[2];
PGSt_double longitude[2];
PGSt_integer numVertPix;
PGSt_integer numHorizPix;
PGSt_integer pixByte;
PGSt_integer totalNumPixels;
int16 *qualityData =NULL;
   /*Some initialization.  Initializing resolutions and layers for PGS_DEM functionality. */
resolutionList[0]= PGSd_DEM_30ARC;
numResolutions = 1;
layerList[0] = PGSd_DEM_ELEV;
```

```
numLayers = 1;
latitude[0] = 40.;
longitude[0] = -100.;
latitude[1] = 38.;
longitude[1] = -97.;
  /*Open the resolution and data layer*/
status = PGS_DEM_Open(resolutionList, numResolutions, layerList,
                                    numLayers);
    if(status != PGS_S_SUCCESS)
    {
          /*ERROR intializing*/
          printf("PGS_DEM_Open: error initializing\n");
    }
    else
    {
          printf("PGS_DEM_Open: Successful Open\n");
    }
    status = PGS_DEM_GetSize(resolutionList[0], PGSd_DEM_GEOID,
                                      PGSd_DEM_DEGREE, latitude, longitude,
                                      &numVertPix, &numHorizPix, &pixByte);
    if(status != PGS_S_SUCCESS)
    {
          /*ERROR with GetSize*/
      printf("PGS_DEM_GetSize: error-- %d\n", status);
    }
    else
    {
          /*print the size of region*/
          printf("PGS_DEM_GetSize: PGSd_DEM_GEOID\n");
          printf("number of bytes in one pixel is %d\n", pixByte);
```

```
        printf("number of pixels vertically spanning region %d\n",
        numVertPix);
        printf("number of pixels horizontally spanning region %d\n",
        numHorizPix);
    }
    /* allocate enough space for qualityData */
    totalNumPixels = numVertPix * numHorizPix;
    qualityData = calloc(totalNumPixels, pixByte);
    if (qualityData == NULL)
    {
        /*error callocing*/
        printf("error callocing\n");
    }
    /* Get Quality Data */
    status = PGS_DEM_GetQualityData(resolutionList[0],
PGSd_DEM_ GEOID, PGSd_DEM_DEGREE, latitude, longitude,
 (void *)qualityData);
    if (status != PGS_S_SUCCESS)
    {
        printf("error: PGS_DEM_GetQualityData\n");
    }
    else
    {
        printf("extracted quality data:using PGS_DEM_GetQualityData\n");
    }
    status = PGS_DEM_Close(resolutionList, numResolutions, layerList,
                numLayers);
    if (status != PGS_S_SUCCESS)
    {
        /*ERROR DE-INITIALIZING*/
    printf("Error closing DEM session.\n");
```

}

FORTRAN: TBD

**NOTES:** All the 30 arc second, 3 arc second, and 90 arc second DEM data are referenced vertically to mean sea level, which is approximated by the geoid. The numbers for geoid that one can extract using PGS_DEM_GetQualityData, as shown in the example, are added to the DEM value to make the height relative to the WGS84 ellipsoid. Thus in order to get height relative to the WGS84 ellipsoid one calls first PGS_DEM_GetPoint (see example for the function PGS_DEM_GetPoint) to retrieve the elevation data with respect to the mean sea level. The subsequent call to PGS_DEM_GetQualityData, as shown in the example, will retrieve geoid data. Then these values are added together to give the height relative to the WGS84 ellipsoid.

qualityField:

For example, one could query the information on either the data source, the quality metric, or the geoid which corresponds to the flags PGSd_DEM_SOURCE, PGSd_DEM_HORIZONTAL_ACCURACY, PGSd_DEM_VERTICAL_ACCURACY, and PGSd_DEM_GEOID respectively.

qualityData - Followings are the data types and values for quality fields:

Source: Data type is 1 byte integer

code 0-8:

0 - no data (ocean)

1 - Digital Terrain Elevation Data (DTED)

2 - Digital Chart of the World (DCW)

3 - USGS 1-degree DEM's

4 - Army Map Service 1:1,000,000-scale maps

5 - International Map of the World 1:1,000,000-scale map

6 - Peru 1:1,000,000-scale map

7 - New Zealand DEM

8 - Antarctic Digital Database (ADD)

Geoid: Data type is 2 byte integer

Data range is -101 to 75 Meters. Add numbers to Mean Sea Level to achieve WGS84 Geoid.

Method: Data type is 1 byte integer.

accuracy calculation method - code 0-5:

0 - no data (ocean)

1 - accuracy from source DEM metadata

2 - vertical accuracy calculated by comparison with higher resolution DEM; horizontal accuracy from source product specification

3 - accuracy from source DEM product specification

4 - vertical accuracy estimated from contour interval of source;

horizontal accuracy estimated from map scale of source

5 - not calculated

1 is used for DTED.

2 is used for DCW.

3 is used for USGS DEM's.

4 is used for cartographic sources (sources 4, 5, 6, and 7 in source data).

5 is used for Antarctica (where the wide range of contour intervals and map scales in the ADD makes it unreasonable to give a reliable estimate).

Hoizontal Accuracy: Data type is 2 byte integer.

absolute horizontal accuracy: RMSE in meters

-9999 = no data (ocean)

9999 = unknown

Vertical Accuracy: Data type is 2 byte integer.

absolute vertical accuracy: RMSE in meters

-9999 = no data (ocean)

9999 = unknown

**REQUIREMENTS:** PGSTK–0946

# Return Size of Specified DEM Region

**NAME:** **PGS_DEM_GetSize()**

**SYNOPSIS:**

C:

```
PGSt_SMF_status
PGS_DEM_GetSize(
PGSt_DEM_Tag  resolution,
PGSt_integer  field,
PGSt_integer positionCode,
PGSt_double latitude[2],
PGSt_double longitude[2],
PGSt_integer *numPixVertical,
PGSt_integer *numPixHorizontal,
PGSt_integer *sizeDataType)
```

FORTRAN:

```
#include <PGS_SMF.f>
#include <PGS_DEM.f>
#include <PGS_DEM_14.f>
#include <PGS_MEM_7.f>
integer function pgs_dem_getsize(resolution, field, positionCode, latitude,
longitude, numPixVertical, numPixHorizontal, sizeDataType)
integer          resolution
integer          field
integer positionCode
double precision        latitude(2)
double precision        longitude(2)
integer numPixVertical
integer numPixHorizontal
integer sizeDataType
```

**DESCRIPTION:** This tool determines the size of a rectangular region defined by the latitudes and longitudes of its upper left and lower right corners. This tool is meant to facilitate the user's ability to allocate appropriate space for the data returned by PGS_DEM_GetRegion and PGS_DEM_GetQualityData. Use of this tool can prevent core dumps and other errors due to improper allocation of memory.

**INPUTS:** resolution - the resolution tag for a particular data set. See Notes to PGS_DEM_DataPresent().

field - either a mask or a qualityField flag. See Notes.

positionCode - flag indicating the format of the position inputs, pntLatitude and pntLongitude. See Notes to PGS_DEM_DataPresent().

latitude[2] and longitude [2] - the latitude and longitude of the points of interest. See Notes to PGS_DEM_SortModels().

**OUTPUTS:** numPixVertical - a pointer to the number of pixels spanning the vertical extent of the region

numPixHorizontal - a pointer to the number of pixels spanning the horizontal extent of the region

sizeDataType - a pointer to the size of an individual pixel of data, in bytes

**RETURNS:** PGS_S_SUCCESS - success

PGSDEM_E_IMPROPER_TAG - Error, improper resolution tag(s)

PGSDEM_E_CANNOT_ACCESS_DATA - Error, cannot access the data set

**EXAMPLES:**

C:

```
PGSt_SMF_status status;

PGSt_integer resolution;

PGSt_integer layer;

PGSt_double latitude[2];

PGSt_double longitude[2];

PGSt_integer numVertPix;

PGSt_integer numHorizPix;

PGSt_integer pixByte;

/* initialize resolution and layer */
```

```
                  resolution = PGSd_DEM_30ARC;

                  layer = PGSd_DEM_ELEV;

/*initialize location of region.  In this case, position is in signed decimal
                  degrees */

                  latitude[0] = 4.0;

                  longitude[0] = 112.0;

                  latitude[1] =  -3.0;

                  longitude[1] = 115.5;

                  status = PGS_DEM_GetSize(resolution, layer, PGSd_DEM_DEGREE,
                  latitude, longitude, &numVertPix, &numHorizPix, &pixByte);

                  if(status != PGS_S_SUCCESS)

                  {

/* Do some error handling ...*/

                  ....
```

FORTRAN:

```
                  integer resolution

                  integer layer

                  integer status

                  double precision latitude(2)

                  double precision longitude(2)

                  integer numVertPix

                  integer numHorizPix

                  integer pixByte


C   **initialize resolution and layer

                  resolution = PGSd_DEM_30ARC

                  layer = PGSd_DEM_ELEV

C   **initialize location of region.  In this case, position is in signed
                  decimal degrees

                  latitude(1) = 4.0
```

```
                longitude(1) = 112.0

                latitude(2) =  -3.0

                longitude(2) = 115.5

                status = PGS_DEM_GetSize(resolution, layer,

     1          PGSd_DEM_DEGREE, latitude, longitude, numVertPix,

     1           numHorizPix, pixByte)

                if(status .NE. PGS_S_SUCCESS) then

C  ** Do some error handling ...**
```

**NOTES:** field:

This indicates the layer attribute or field of the quality assurance layer over which the region is "sized". For ECS Deliveries B.0, the layers that may be inputted are elevation, standard deviation of elevation, water/land, slope gradient, standard deviation of slope gradient, aspect , data source, quality metric, and geoid which correspond to the flags PGSd_DEM_ELEV, PGSd_DEM_STDEV_ELEV, PGSd_DEM_WATER_LAND, PGSd_DEM_SLOPE, PGSd_DEM_STDEV_SLOPE, PGSd_DEM_ASPECT, PGSd_DEM_SOURCE PGSd_DEM__HORIZONTAL_ACCURACY, PGSd_DEM_VERTICAL_ACCURACY, and PGSd_DEM_GEOID, respectively. The layers that will be available in the future are: topographical obscuration (PGSd_DEM_TOP_OBSC), and topographical shadow (PGS_DEM_TOP_SHAD). Note that for 90 arc second data the only available layer is elevation.

**REQUIREMENTS:** PGSTK–0947

### 6.3.2 Ancillary Data Tools

#### 6.3.2.1 Introduction

There will be a large number of ancillary data files used in ECS instrument processing. The tools in this section address files already identified at this writing.

Users could utilize language standard input/output functions or the HDF tools to access the ancillary data. However, a suite of higher level tools is required for the following reasons:

a. to enable data from locations specified by the user to be returned to the user thus avoiding having to know the internal structure of the file.

b. to shield the user from having to know details of parameter source or source format or to track changes in either, although source changes will be agreed upon with the user.

c. to provide for certain additional manipulations of extracted data.

For this final point (c), only those data sets that have been specifically identified as requiring particular manipulations will be serviced; i.e., the ancillary tools do not intend to provide a general manipulation service for all types of data. However, the tools that extract from location (a) will be sufficiently generic to allow additional data sets of a similar type to be used.

# Access the Digital Chart of the World Database

**NAME:** **PGS_AA_dcw( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_Status
PGS_AA_dcw (char iparms[][100],   coverage name—PO
        PGSt_integer nParms,      number of coverages
        PGSt_double longitude[],  longitude of point(s)
        PGSt_double latitude[],   latitude of point(s)
        PGSt_integer npoints,     number of points
        void *results)            result of search
```

FORTRAN:

```
include'PGS_AA_10.f'

integer function
PGS_AA_dcw(parms, nParms, latitude, longitude, npoints, results)
        character*99  iparms(*),
        integer       nParms,
        double        latitude(*)
```

**DESCRIPTION:** This routine receives either a single point or an array of location points and navigates the DCW database in order to find the coverage that the user supplies as parm. Once the coverage is identified, the database path is updated, with each file and table identified, until the table containing the locational information is located. Once this table is found, the table is opened and the result for a latitude/longitude is extracted and returned in results.

```
[start]
PERFORM    PGS_AA_dcw_Parm
PERFORM    PGS_AA_dcw_Intile
PERFORM    PGS_AA_dcw_Inface
PERFORM    PGS_AA_dcw_Feature
PERFORM    return PGS_S_SUCCESS

[end]
```

**INPUTS:**

*Table 6-138. PGS_AA_dcw Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| parms | parameter wanted | N/A | N/A | N/A |
| nParms | number of parameters | N/A | 1 | 1 |
| latitude | latitude location | degrees | -90.0 | 90.0 |
| longitude | longitude location | degrees | -180.0 | 180.0 |
| npoints | number of points | N/A | 0 | Unlimited |

**OUTPUTS:**

*Table 6-139. PGS_AA_dcw Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| results | extracted parameter | char | N/A | N/A |

**RETURNS:**

*Table 6-140. PGS_AA_dcw Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSAA_E_DCW_ERROR | Error in extracting value required |
| PGSAA_W_DCW_NODATA | No data at that point in data base |

The following errors are reported to the error log

PGSAA_E_CANT_FIND_PARM
PGSAA_E_CANT_GET_CONTINENT_PATH
PGSAA_E_CANT_GET_TILE_DIR
PGSAA_E_CANT_GET_POINT_IN_FACE
PGSAA_E_CANT_GET_POINT_INFO

**EXAMPLES:**

C:

```
#include <PGS_AA.h>

PGSt_double latitude[2] = {-9.29, -25.34};
PGSt_double longitude[2] = {110.3, 30.9};
PGSt_integer results[2];
char parm[PGSd_AA_MAXNOCACHES][100] = {"po"};

ret_status = PGS_AA_dcw(parm, 1, longitude, latitude, 2,
                        results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           PGS_AA_dcw
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms(2)
double            latitude(2)
double            longitude(2)
integer           npoints(2)
integer           result(2)
parms(1)= "po"
nParms = 2
latitude = -9.29, -25.34
longitude = 110.3, 30.9
npoints = 2

call pgs_aa_dcw(parms, nParms, longitude, latitude, npoints,
                results)
```

**NOTES:** For further details of the background to this tool see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide).

IMPORTANT: The PGS_AA_dcw code calls a number of library modules, which carry out such actions as mallocing memory for files, opening files, opening tables, reading tables, extracting information from tables and closing tables. These library modules are detailed in the DCW format specification and the associated vector product format (VPF) library software.

NOTE: Precision of latitude and longitude is machine specific, not data–base specific.

**REQUIREMENTS:** PGSTK–0840, PGSTK–0870, PGSTK–1360, PGSTK–1362

# Access Available Data from a Set of Standard Digital Elevation Models (DEMs)

**NAME:** **PGS_AA_dem( )**

**SYNOPSIS:**

C:

```
#include "PGS_AA.h"

PGSt_SMF_status
PGS_AA_dem(char parms[][100],
        PGSt_integer        nParms,
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_integer        versionFlag[],
        PGSt_integer        nPoints,
        PGSt_PC_Logical     fileId,
        PGSt_integer        operation,
        void                *results)
```

FORTRAN:

```
include"PGS_AA_10.f"
include"PGS_AA.f"

integer function
pgs_aa_dem (parms, nparms, latitude, longitude,
            versionflag, npoints, fileId, operation, results)
        character*99        parms(*)
        integer             nParms
        double precision    latitude(*)
        double precision    longitude(*)
        integer             versionflag(*)
        integer             npoints
        integer             fileId
        integer             operation
        'user specified'    results (see Notes)
```

**DESCRIPTION:** This routine provides the interface to retrieve DEM values from the gridded data set.

**INPUTS:**

*Table 6-141. PGS_AA_dem Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| parms | parameter names requested | see notes | | |
| nParms | number of parms | none | 1 | #defined |
| latitude | latitude(s) of the requested point | degrees | -90.00 | 90.00 |
| longitude | longitude(s) of the requested point | degrees | -180.00 | 180.00 |
| nPoints | no. of points requested | none | 1 | variable |
| fileId | logical file number | none | variable | variable |
| operation | defines user required | none | 1 | variable |

**OUTPUTS:**

*Table 6-142. PGS_AA_dem Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| versionFlag | indicates tile location for a point (see notes) | see notes | 1 | variable |
| results | results | see notes | | |

**RETURNS:**

*Table 6-143. PGS_AA_dem Returns*

| Returns | Descriptions |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSAA_E_NPOINTSINVALID | Number of points invalid |
| PGSAA_E_TILE_STATUS | Could not establish tile status of the DEM file |
| PGSAA_E_2DGEO | Error returned from PGS_AA_2Dgeo |
| PGSAA_E_SUPPORTID | Could not establish support file id |
| PGSAA_E_MINMAX | Could not establish min/max range for the DEM |
| PGSAA_E_DATATYPE | Could not establish parameter datatype |
| PGSAA_E_UNKNOWN_DATATYPE | DEM datafile datatype is unknown |

**EXAMPLES:**

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] = { "USAelevation" };
long nParms = 1;
PGSt_double latitude[MAX_POINTS] = {51.5, 51.23666,
                                    50.973333} ;
PGSt_double longitude[MAX_POINTS] = {0.1666666,0.3832,
                                     0.5999};

PGSt_integer  versionFlag[MAX_POINTS];

PGSt_integer nPoints = 3;
long fileId = 210;
long version = 0;
long operation = 1;
short results[3];
retStatus = PGS_AA_2Dgeo(parms, nParms, latitude, longitude,
                         versionFlag, nPoints, fileId,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

parms(PGSd_AA_MAXNOCACHES)
integer                 pgs_aa_dem
integer                 versionFlag(300)
integer                 nParm
double precision        latitude(300)
double precision        longitude(300)
integer                 fileId
integer                 nPoints
integer                 version
integer                 operation
integer                 results(300)
integer                 retStatus
parms(1)= "USAelevation"
nParms = 1
fileId = 202
operation = 1
nPoints = 300
do 10 i = 1, 300
.
.
```

```
                latitude(i) = calculated_user_lat
                longitude(i) = calculated_user_lon
                .
                .
10    continue
                retStatus = pgs_aa_dem( parms, nparms, latitude,
                2           longitude, versionFlag, npoints,
                1             fileId, operation, results )
```

**NOTES:** The added facility that differentiates this tool from its sister tool PGS_AA_2Dgeo is that this routine can handle tiled data sets by selecting from geographically separated tiles. Some of the DEM datafiles can be very large files and are necessarily tiled into smaller files to avoid memory problems.

Also this routine processes all input point data and returns a warning if some of the input points were found to be out of range. In such an event user can examine versionFlag[] to locate the offending points. For such points the corresponding location in versionFlag would contain a value PGSd_AA_OUT_OF_RANGE, e.g.,

if latitude[3] and longitude[3] is the offending point then

versionFlag[3] = PGSd_AA_OUT_OF_RANGE.

For other points the versionFlag[] would actually contain the number of the tile where the point was located.

For the details of DEM datafiles the user is referred to appendix D.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The results buffer holds the final output sent back to the user. It can hold data of 4 types (long, short, float, double).

For more details the user is referred to information regarding PGS_AA_2Dgeo.

Users MUST be aware of the amount of disk space required by the number of calls to the tool (where calls demand the ingestion of separate physical files), and in doing so not exceed the capacity of the machine they are working on.

**DEC—users**

DEC users should be aware that for some of the product files a DEC version (e.g., etop05.dat_dec) is supplied. The user should use these instead of the normal files. This is for backward compatibility with the PGS_AA_2Dgeo tool. For the rest of the data files there is an inbuilt facility to swap the bytes. For these files there is a flag 'swapBytes = yes' in the support file. This flag is set to 'no' for the data files with 'dec' versions. Another issue that the user should be aware of is that DEC

represents 'long' datatype as 8 bytes long. Therefore, if there is a datafile created on a different platform (most other platforms represent 'long' as 4 bytes), then that file must be converted first to be used on the DEC. Conversion should simply be reading the file as 'int' (4 bytes) and writing it out as 'long' (8 bytes) on the DEC. To take care of byteswapping the support file for such datafile should contain a flag 'swapBytes = yes'.

**REQUIREMENTS:** PGSTK–0840, PGSTK–0980

## Extract String Parameter from Parameter=Value Formatted File

**NAME:** **PGS_AA_PeVA_string( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_string(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        char                *value[] )
```

FORTRAN:

```
include"PGS_AA_10.f"
include"PGS_AA.f"

integer function
pgs_aa_peva_string( pevLogical, parameter, value )
        integer         pevLogical
        character*(*)   parameter
        character*(*)   value
```

**DESCRIPTION:** This routine returns the value associated with a string type parameter from the given file.

**INPUTS:**

***Table 6-144. PGS_AA_PeVA_string Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| pevLogical | file logical for file to be accessed | see notes | | |
| parameter | name of parameter to be retrieved | see notes | | |

**OUTPUTS:**

***Table 6-145. PGS_AA_PeVA_string Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| value | value associated with retrieved parameter | see notes | | |

**RETURNS:**

***Table 6-146. PGS_AA_PeVA_string Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSAA_E_PEV_ERROR | Error in extracting the required value |

The following errors are reported to the error log

PGSAA_E_CANT_GET_FILE_ID
PGSAA_E_CANT_OPEN_INPUT_FILE
PGSAA_E_AGG_CANT_BE_INSERTED
PGSAA_E_READLABEL_PARSE_ERROR
PGSAA_E_PARAMETER_INVALID
PGSAA_E_FIRST_NODE_NOT_FOUND

**EXAMPLE:**

C:

```
#include    <PGS_AA.h>
#define     MAX_STRING 30

PGSt_SMF_status retStatus;
char        *myStringValue[MAX_STRING]= {"          "
                                          "          "
                                          "
"};

ret_status = PGS_AA_PeVA_string(MY_PEV_FILE,
                                "MY_STRING_PARAMETER",
                                 myStringValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_string
integer           pevLogical, return
character*30      parameter
character*20      value
```

```
pevLogical = 876
parameter = "dataType"

return = pgs_aa_peva_string( pevLogical, parameter, value )
```

**NOTES:** The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is also a string as returned from the data file identified by the logical. For

**REQUIREMENTS:** PGSTK–1365

## Extract Real Parameter from Parameter = Value Formatted File

**NAME:** **PGS_AA_PeVA_real( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_real(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        PGSt_double         *value )
```

FORTRAN:

```
include'PGS_AA_10.f'
include'PGS_AA.f'

integer function
pgs_aa_peva_real( pevLogical, parameter, value )
        integer             pevLogical
        character*(*)       parameter
        double precision    value
```

**DESCRIPTION:** This routine returns the value associated with a string type parameter from the given file.

**INPUTS:**

***Table 6-147. PGS_AA_PeVA_real Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| pevLogical | file logical for file to be accessed | see notes | | |
| parameter | name of parameter to be retrieved | see notes | | |

**OUTPUTS:**

***Table 6-148. PGS_AA_PeVA_real Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| value | value associated with retrieved parameter | see notes | | |

**RETURNS:**

*Table 6-149. PGS_AA_PeVA_real Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSAA_E_PEV_ERROR | Error in extracting the required value |

The following errors are reported to the error log

PGSCUC_E_CANT_GET_FILE_ID
PGSCUC_E_CANT_OPEN_INPUT_FILE
PGSCUC_E_AGG_CANT_BE_INSERTED
PGSCUC_E_READLABEL_PARSE_ERROR
PGSCUC_E_PARAMETER_INVALID
PGSCUC_E_FIRST_NODE_NOT_FOUND

**EXAMPLE:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status   retStatus;
PGSt_double       myRealValue[10];

ret_status = PGS_AA_PeVA_real(MY_PEV_FILE,
                              "MY_STRING_PARAMETER",
                               &myRealValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_real
integer           pevLogical, return
character*30      parameter
double precision  value
pevLogical = 876
parameter = "maxLat"

return = pgs_aa_peva_real( pevLogical, parameter, value )
```

**NOTES:** The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is a real as returned from the data file identified by the logical.

**REQUIREMENTS:** PGSTK–1365

## Extract Integer Parameter from Parameter = Value Formatted File

**NAME:** **PGS_AA_PeVA_integer( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_PeVA_integer(
        PGSt_uinteger       pevLogical,
        char                *parameter,
        PGSt)integer        *value)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_peva_integer( pevLogical, parameter, value )
        integer         pevLogical
        character*(*)   parameter
        integer         value
```

**DESCRIPTION:** This routine returns the value associated with a string type parameter from the given file.

**INPUTS:**

***Table 6-150. PGS_AA_PeVA_integer Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| pevLogical | file logical for file to be accessed | see notes | | |
| parameter | name of parameter to be retrieved | see notes | | |

**OUTPUTS:**

***Table 6-151. PGS_AA_PeVA_integer Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| value | value associated with retrieved parameter | see notes | | |

**RETURNS:**

***Table 6-152. PGS_AA_PeVA_integer Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSAA_E_PEV_ERROR | Error in extracting the required value |

The following errors are reported to the error log

PGSCUC_E_CANT_GET_FILE_ID
PGSCUC_E_CANT_OPEN_INPUT_FILE
PGSCUC_E_AGG_CANT_BE_INSERTED
PGSCUC_E_READLABEL_PARSE_ERROR
PGSCUC_E_PARAMETER_INVALID
PGSCUC_E_FIRST_NODE_NOT_FOUND

**EXAMPLE:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status   retStatus;
PGSt_integer      myIntValue;

ret_status = PGS_AA_PeVA_integer(MY_PEV_FILE,
                                 "MY_STRING_PARAMETER",
                                  &myIntValue);

if (ret_status != PGS_S_SUCCESS)
{
      signal ERROR
}
```

FORTRAN:

```
implicit none

include           'PGS_AA.f'
include           'PGS_AA_10.f'

integer           pgs_aa_peva_integer
integer           pevLogical, return
character*30      parameter
integer*(*)       value
pevLogical = 876
parameter = "size"

return = pgs_aa_peva_real( pevLogical, parameter, value )
```

**NOTES:** The logical is an integer whose value is supplied through the PC tools. The parameter is a data set dependent character string and the value is a real as returned from the data file identified by the logical.

**REQUIREMENTS:** PGSTK–1365

## Extract Data from Gridded Data Sets by Geographic Location

**NAME:** **PGS_AA_2Dgeo( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_2Dgeo ( char  iparms[][100],
        PGSt_integer  nParms,
        PGSt_double   latitude[],
        PGSt_double   longitude[],
        PGSt_integer  nPoints,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void          *results);
```

FORTRAN:

```
include'PGS_AA_10.f'
include'PGS_AA.f'

integer function
pgs_aa_2dgeo( parms, nparms, latitude, longitude, fileId,
              version, operation, results )
        character*99        parms(*)
        integer             nParms
        real*8              latitude(*)
        real*8              longitude(*)
        integer             fileId
        integer             version
        integer             operation
        'user specified'    results (see Notes)
```

**DESCRIPTION**: The user specifies a parameter, a fileId and version from which the data are to be extracted using the geographic coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that extracts gridded data by geographic location.

```
[start]
PERFORM    PGS_AA_Map
PERFORM    PGS_AA_GetSupp    to get support data
DO         allocate memory to parmBuffer using
```

```
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
DO          set tool used to 2
PERFORM     PGS_AA_GEOGrid
[end]
```

**INPUTS**:

***Table 6-153. PGS_AA_2Dgeo Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| parms | parameter names | see notes | requested | |
| nParms | number of parms | none | 1 | #defined |
| latitude | latitude(s) of the requested point | degrees | -90.00 | 90.00 |
| longitude | longitude(s) of the requested point | degrees | -180.00 | 180.00 |
| nPoints | no. of points requested | none | 1 | variable |
| fileId | logical file number | none | variable | variable |
| version | version of dynamic file | none | 1 | variable |
| operation | defines user required | none | 1 | variable |

**OUTPUTS**:

***Table 6-154. PGS_AA_2Dgeo Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| results | results | see notes | | |

**RETURNS**:

*Table 6-155. PGS_AA_2Dgeo Returns*

| To User/Log File | Return | Description |
|---|---|---|
| u | PGSAA_E_GEOERRO | Error in GEO extraction |
| l | PGSAA_E_AUTOOPERATION | Error in executing autoOperation |
| l | PGSAA_E_AUTOOPERATIONUNSET | No autoOperation found in support file |
| l | PGSAA_E_OPERATION | Error in executing operation |
| l | PGSAA_E_OPERATIONUNSET | Operation not set by user |
| ul | PGSAA_E_GEOTOSTRUCT | Failure in calculation of structure from lat/lon |
| ul | PGSAA_E_UNIDENTIFIEDTYPE | Type cannot be identified, results failure |
| u | PGSAA_E_SUPPORTFILE | Support or format files inaccessible |
| ul | PGSAA_E_PARMSNOTFOUND | Parameter(s) not found in the support support file |
| l | PGSAA_E_DATARATEUNSET | dataRate attribute unset in support file |
| ul | PGSAA_E_PARMSFROMANYFILES | Parameters requested from more than one physical file |
| ul | PGSAA_E_INVALIDNOPARMS | No of parms incorrect |
| ul | PGSAA_E_BADSUPPSUPPORT | Tool support file is corrupted or incomplete |
| ul | PGSAA_E_CANTFINDFILE | Format of input data file inaccessible |
| u | PGSAA_E_FFERROR | A freeform error has occured |
| l | PGSAA_E_FFDBIN | Failure in Freeform make_dbin function |
| l | PGSAA_E_FFDBSET | Failure in Freeform db_set function |
| l | PGSAA_E_FFDBEVENTS | Failure in Freeform db_events function |
| ul | PGSAA_E_MALLOC | Failure to malloc |
| u | PGSAA_E_PEV_ERROR | An error has occurred in the PeVA tool |
| l | PGSAA_E_PEV_XS_SUPPFILES | Too many PeVA files open, increase MAXFILES |
| l | PGSAA_E_CANT_GET_VALUE | Unable to extract value from dbin |
| l | PGSAA_E_GETDBIN | Error in PeVA tool obtaining dbin |
| u | PGSAA_E_GETSUPP | An error was detected while extracting support data |
| l | PGSAA_E_POSITION_CALC_FAILURE | The position in the parmBuffer of the requested values was miscalculated |
| u | PGSAA_E_TWOD_READ_ERROR | Function failure to read parameter values from buffer |
| l | PGSAA_E_EXTRACTORESULTSERROR | Failure to transfer selected values from parmBuffer to results |
| ul | PGSAA_E_OUTOFRANGE | Input values out of data set range |

**EXAMPLE:**

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] = {
                                        "etop05SeaLevelElevM" };
long nParms = 1;

PGSt_double latitude[] = {51.5, 51.23666, 50.973333} ;
PGSt_double longitude[] = {0.1666666,0.3832,  0.5999};

PGSt_integer nPoints = 3;

PGSt_integer fileId = 10955;

PGSt_integer version = 1;

PGSt_integer operation = 1;

short results[3];

retStatus = PGS_AA_2Dgeo(parms, nParms, latitude, longitude,
                         nPoints, fileId, version,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_2dgeo
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
real*8            latitude(300)
real*8            longitude(300)
integer           fileId
integer           nPoints
integer           version
integer           operation
integer           results(300)
parms(1)= "fnocMod"
nParms = 1
fileId = 10965
operation = 1
version = 1
nPoints = 300
do 10 i = 1, 300
.
.
```

```
            latitude(i) = calculated_user_lat
            longitude(i) = calculated_user_lon
            .
            .
10             continue

            call pgs_aa_2dgeo( parms, nparms, latitude, longitude,
                               fileId, version, operation, results )
```

**NOTES:** For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The parmBuffer input is a memory buffer holding whatever data is extracted form the data set requested by the user. The results buffer is similar although holds the final output sent back to the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

**REQUIREMENTS:** PGSTK–0840, PGSTK–0931, PGSTK–0980, PGSTK–1030, PGSTK–1362

## Extract Data from Gridded Data Sets by Geographic Location

**NAME:** **PGS_AA_3Dgeo( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_3Dgeo ( char iparms[][100],
        PGSt_integer  nParms,
        PGSt_double   latitude[],
        PGSt_double   longitude[],
        PGSt_integer  height[],
        PGSt_integer  nPoints,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void          *results);
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

        character*99        iparms(PGSd_AA_MAXNOCACHES)
        integer             nParms
        real*8              latitude(*)
        real*8              longitude(*)
        integer             height(*)
        integer             fileId
        integer             version
        integer             operation
        'user specified'    results (see Notes)

integer function
pgs_aa_3dread( parms, nparms, latitude, longitude,
                height, fileId, version, operation, results )
```

**DESCRIPTION:** The user specifies a parameter, a fileId and version from which the data are to be extracted using the geographic coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that extract gridded data by geographic location.

```
[start]
PERFORM    PGS_AA_Map
PERFORM    PGS_AA_GetSupp    to get support data
DO         allocate memory to parmBuffer using
```

```
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
DO          set tool used to 3
PERFORM     PGS_AA_GEOGrid
[end]
```

**INPUTS:**

***Table 6-156. PGS_AA_3Dgeo Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| iparms | parameter names requested | see notes | | |
| nParms | number of parms | none | 1 | 4 |
| latitude | latitude(s) of the requested point | degrees | -90.00 | 90.00 |
| longitude | longitude(s) of the requested point | degrees | -180.00 | 180.00 |
| height | height of the requested point | none | 1 | variable |
| nPoints | no. of points requested | none | 1 | variable |
| fileId | logical file number | none | variable | variable |
| version | version of dynamic file | none | 1 | variable |
| operation | defines user required | none | 1 | variable |

**OUTPUTS:**

***Table 6-157. PGS_AA_3Dgeo Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| results | results | see notes | | |

**RETURNS**:

***Table 6-158. PGS_AA_3Dgeo Returns (1 of 2)***

| To User/Log File | Return | Description |
|---|---|---|
| u | PGSAA_E_GEOERROR | Error in GEO extraction |
| l | PGSAA_E_AUTOOPERATION | Error in executing autoOperation |
| l | PGSAA_E_AUTOOPERATIONUNSET | No autoOperation found in support file |
| l | PGSAA_E_OPERATION | Error in executing operation |
| l | PGSAA_E_OPERATIONUNSET | Operation not set by user |
| ul | PGSAA_E_GEOTOSTRUCT | Failure in calculation of structure from lat/lon |
| ul | PGSAA_E_UNIDENTIFIEDTYPE | Type cannot be identified, results failure |
| u | PGSAA_E_SUPPORTFILE | Support or format files inaccessible |

*Table 6-158. PGS_AA_3Dgeo Returns (2 of 2)*

| To User/Log File | Return | Description |
|---|---|---|
| ul | PGSAA_E_PARMSNOTFOUND | Parameter(s) not found in the support support file |
| l | PGSAA_E_DATARATEUNSET | dataRate attribute unset in support file |
| ul | PGSAA_E_PARMSFROMANYFILES | Parameters requested from more than one physical file |
| ul | PGSAA_E_INVALIDNOPARMS | No of parms incorrect |
| ul | PGSAA_E_BADSUPPSUPPORT | Tool support file is corrupted or incomplete |
| ul | PGSAA_E_CANTFINDFILE | Format of input data file inaccessible |
| u | PGSAA_E_FFERROR | A freeform error has occured |
| l | PGSAA_E_FFDBIN | Failure in Freeform make_dbin function |
| l | PGSAA_E_FFDBSET | Failure in Freeform db_set function |
| l | PGSAA_E_FFDBEVENTS | Failure in Freeform db_events function |
| ul | PGSAA_E_MALLOC | Failure to malloc |
| u | PGSAA_E_PEV_ERROR | An error has occurred in the PeVA tool |
| l | PGSAA_E_PEV_XS_SUPPFILES | Too many PeVA files open, increase MAXFILES |
| l | PGSAA_E_CANT_GET_VALUE | Unable to extract value from dbin |
| l | PGSAA_E_GETDBIN | Error in PeVA tool obtaining dbin |
| u | PGSAA_E_GETSUPP | An error was detected while extracting support data |
| l | PGSAA_E_POSITION_CALC_FAILURE | The position in the parmBuffer of the requested values was miscalculated |
| u | PGSAA_E_THREED_READ_ERROR | Function failure to read parameter values from buffer |
| l | PGSAA_E_EXTRACTORESULTSERROR | Failure to transfer selected values from parmBuffer to results |
| ul | PGSAA_E_OUTOFRANGE | Input values out of data set range |

## EXAMPLE:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

char parms[PGSd_AA_MAXNOCACHES][100] =
                        {"nmcRucSigPres","nmcRucSigPot"};
long nParms = 2;

PGSt_double latitude[] = {51.5, 51.23666, 50.973333} ;
PGSt_double longitude[] = {0.1666666, 0.3832, 0.5999};
long height[] = {1,2,1};
PGSt_integer nPoints = 3;

long fileId = 10972;
```

```
long version = 1;

long operation = 2;

short results[3][2];

retStatus = PGS_AA_3Dgeo(parms, nParms, latitude, longitude,
                         height, nPoints, fileId, version,
                         operation, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_3dgeo
character*99      iparms(PGSd_AA_MAXNOCACHES)
integer           nParms
real*8            latitude(300)
real*8            longitude(300)
integer           height(300)
integer           fileId
integer           nPoints
integer           version
integer           operation
integer           results(2,300)
parms(1)= "nmcRucSigPres"

parms(2)= "nmcRucSigPot"
nParms= 2
fileId = 10972
operation = 2
version = 1
nPoints = 300
do 10 i = 1, 300
.
.
latitude(i) = calculated_user_lat
longitude(i) = calculated_user_lon
height(i) = calculate_user_height
.
.

10      continue

call pgs_aa_3dgeo( iparms, nparms,latitude, longitude,
                   height, fileId, version, operation,
                   results )
```

**NOTES:** For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

Height or the z dimension is a layer number in the file and is data set dependent.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

**REQUIREMENTS:** PGSTK–0931, PGSTK–0840, PGSTK–1362

# Extract Data from Gridded Data Sets by File Structure

**NAME:** **PGS_AA_2DRead( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_2DRead(
        char iparms[][100],
        PGSt_integer  nParms,
        PGSt_integer  xStart,
        PGSt_integer  yStart,
        PGSt_integer  xDim,
        PGSt_integer  yDim,
        PGSt_integer  fileId,
        PGSt_integer  version,
        PGSt_integer  operation,
        void           *results)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_2dread( parms, nparms, xStart, yStart, xDim, yDim, fileId,
                version, operation, results )
        character*99          parms(*)
        integer       nParms,
        integer       xStart,
        integer       yStart,
        integer       xDim,
        integer       yDim,
        integer       fileId,
        integer       version,
        integer       operation,
        'user specified' results (see Notes)
```

**DESCRIPTION:** The user specifies a parameter, a fileId and version from which the data are to be extracted using the data structure coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that accepts the arguments and calls PGS_AA_Map, PGS_AA_GetSupp, PGS_AA_FF_Setup and PGS_AA_2DReadGrid. The first 3 of these modules determine the

validity of the call and initialize support and load the identified data into memory. PGS_AA_2DReadGrid performs the extraction requested from the input arguments

```
[start]
PERFORM     PGS_AA_Map
PERFORM     PGS_AA_GetSupp     to get support data
DO          allocate memory to parmBuffer using
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
PERFORM     PGS_AA_2DReadGrid
[end]
```

**INPUTS:**

***Table 6-159. PGS_AA_2DRead Input***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| parms | parameter names requested | see notes | | |
| nParms | number of parms | none | 1 | #defined |
| xStart | the x start point | none | 1 | variable |
| yStart | the y start point | none | 1 | variable |
| xDim | the x dimension | none | 1 | variable |
| yDim | the y dimension | none | 1 | variable |
| fileId | logical file number | none | variable | variable |
| version | version of dynamic file | none | 1 | variable |
| operation | defines user required | none | 1 | variable |

**OUTPUTS:**

***Table 6-160. PGS_AA_2DRead Output***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| results | results | variable | N/A | N/A |

## RETURNS:

*Table 6-161. PGS_AA_2DRead Returns*

| To User/Log File | Return | Description |
|---|---|---|
| u | PGSAA_E_SUPPORTFILE | Support or format files inaccessible |
| ul | PGSAA_E_PARMSNOTFOUND | Parameter(s) not found in the support support file |
| l | PGSAA_E_DATARATEUNSET | dataRate attribute unset in support file |
| ul | PGSAA_E_PARMSFROMANYFILES | Parameters requested from more than one physical file |
| ul | PGSAA_E_INVALIDNOPARMS | No of parms incorrect |
| ul | PGSAA_E_BADSUPPSUPPORT | Tool support file is corrupted or incomplete |
| ul | PGSAA_E_CANTFINDFILE | Format of input data file inaccessible |
| u | PGSAA_E_FFERROR | A freeform error has occured |
| l | PGSAA_E_FFDBIN | Failure in Freeform make_dbin function |
| l | PGSAA_E_FFDBSET | Failure in Freeform db_set function |
| l | PGSAA_E_FFDBEVENTS | Failure in Freeform db_events function |
| ul | PGSAA_E_MALLOC | Failure to malloc |
| u | PGSAA_E_PEV_ERROR | An error has occurred in the PeVA tool |
| l | PGSAA_E_PEV_XS_SUPPFILES | Too many PeVA files open, increase MAXFILES |
| l | PGSAA_E_CANT_GET_VALUE | Unable to extract value from dbin |
| l | PGSAA_E_GETDBIN | Error in PeVA tool obtaining dbin |
| u | PGSAA_E_GETSUPP | An error was detected while extracting support data |
| l | PGSAA_E_POSITION_CALC_FAILURE | The position in the parmBuffer of the requested values was miscalculated |
| u | PGSAA_E_TWOD_READ_ERROR | Function failure to read parameter values from buffer |
| l | PGSAA_E_EXTRACTORESULTSERROR | Failure to transfer selected values from parmBuffer to results |
| ul | PGSAA_E_OUTOFRANGE | Input values out of data set range |

## EXAMPLE:

C:

```
#include <PGS_AA.h>
PGSt_SMF_status retStatus;

short results[50][20]
char parm[PGSd_AA_MAXNOCACHES][100] =
                              {"OlsonWorldEcosystems1.3a"};
PGSt_integer      nParms = 1;
PGSt_integer      xStart = 4;
PGSt_integer      yStart = 7;
PGSt_integer      xDim = 20;
```

```
PGSt_integer      yDim = 50;
PGSt_integer      fileId = 10952;

PGSt_integer version = 1;

PGSt_SMF_status = PGS_AA_2DRead (parm, nParms, xStart,
                                 yStart, xDim, yDim, fileId,
                                 version, 1, results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_2dread
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
integer           xStart
integer           yStart
integer           xDim
integer           yDim
integer           fileId
integer           version
integer           operation
PGSt_integer      results(14, 20)
parms(1)= "OlsonWorldEcosystems1.3a"
nParms = 1
yStart = 102
xStart = 205
yDim = 20
xDim = 14
fileId = 10952
operation = 1
version = 1

call pgs_aa_2dread( parms, nparms, xStart, yStart, xDim,
                    yDim, fileId, version, operation,
                    results )
```

**NOTES:** For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

**REQUIREMENTS:** PGSTK–0931, PGSTK–0980, PGSTK–1000, PGSTK–1030, PGSTK–1360, PGSTK–1362

## Extract Data from Gridded Data Sets by File Structure Parameters

**NAME:** **PGS_AA_3DRead( )**

**SYNOPSIS:**

C:

```
#include <PGS_AA.h>

PGSt_SMF_status
PGS_AA_3DRead(
        char iparms[][100],
        PGSt_integer nParms
        PGSt_integer xStart
        PGSt_integer yStart
        PGSt_integer zStart
        PGSt_integer xDim
        PGSt_integer yDim
        PGSt_integer zDim
        PGSt_integer fileId
        PGSt_integer version
        PGSt_integer operation
        void *results)
```

FORTRAN:

```
include'PGS_AA_10.f'
include"PGS_AA.f"

integer function
pgs_aa_3dread( iparms, nparms, xStart, yStart, zStart, xDim, yDim, zDim,
                fileId, version, operation, results )
        character*99            iparms(*)
        integer         nParms,
        integer         xStart,
        integer         yStart,
        integer         zStart,
        integer         xDim,
        integer         yDim,
        integer         zDim,
        integer         fileId,
        integer         version,
        integer         operation,
        'user specified' results (see Notes)
```

**DESCRIPTION:** The user specifies a parameter, a fileId and version from which the data are to be extracted using the file structure coordinates. Since this tool is similar to the other AA geo and Read tools, a single explanation of the interface is provided in Appendix D.3.

The interface to the calling algorithm that accepts the arguments and calls PGS_AA_Map, PGS_AA_GetSupp, PGS_AA_FF_Setup and PGS_AA_3DReadGrid. The first 3 of these modules determine the validity of the call and initialize support and load the identified data into memory. PGS_AA_3DReadGrid performs the extraction requested from the input arguments

```
[start]
PERFORM     PGS_AA_Map
PERFORM     PGS_AA_GetSupp     to get support data
DO          allocate memory to parmBuffer using
        totalParmMemoryCache
PERFORM     PGS_AA_FF_Setup
PERFORM     PGS_AA_3DReadGrid
[end]
```

**INPUTS:**

***Table 6-162. PGS_AA_3DRead Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| parms | parameter names requested | see notes | | |
| nParms | number of parms | none | 1 | #defined |
| xStart | the x start point | none | 1 | variable |
| yStart | the y start point | none | 1 | variable |
| zStart | he z start point | none | 1 | variable |
| xDim | the x dimension | none | 1 | variable |
| yDim | the y dimension | none | 1 | variable |
| zDim | the z dimension | none | 1 | variable |
| fileId | logical file number | none | variable | variable |
| version | version of dynamic file | none | 1 | variable |
| operation | defines user required | none | 1 | variable |

**OUTPUTS:**

***Table 6-163. PGS_AA_3DRead Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| results | results | see notes | | |

## RETURNS:

*Table 6-164. PGS_AA_3DRead Returns*

| To User/Log File | Return | Description |
|---|---|---|
| u | PGSAA_E_SUPPORTFILE | Support or format files inaccessible |
| ul | PGSAA_E_PARMSNOTFOUND | Parameter(s) not found in the support support file |
| l | PGSAA_E_DATARATEUNSET | dataRate attribute unset in support file |
| ul | PGSAA_E_PARMSFROMANYFILES | Parameters requested from more than one physical file |
| ul | PGSAA_E_INVALIDNOPARMS | No of parms incorrect |
| ul | PGSAA_E_BADSUPPSUPPORT | Tool support file is corrupted or incomplete |
| ul | PGSAA_E_CANTFINDFILE | Format of input data file inaccessible |
| u | PGSAA_E_FFERROR | A freeform error has occured |
| l | PGSAA_E_FFDBIN | Failure in Freeform make_dbin function |
| l | PGSAA_E_FFDBSET | Failure in Freeform db_set function |
| l | PGSAA_E_FFDBEVENTS | Failure in Freeform db_events function |
| ul | PGSAA_E_MALLOC | Failure to malloc |
| u | PGSAA_E_PEV_ERROR | An error has occurred in the PeV tool |
| l | PGSAA_E_PEV_XS_SUPPFILES | Too many PeVA files open, increase MAXFILES |
| l | PGSAA_E_CANT_GET_VALUE | Unable to extract value from dbin |
| l | PGSAA_E_GETDBIN | Error in PeVA tool obtaining dbin |
| u | PGSAA_E_GETSUPP | An error was detected while extracting support data |
| l | PGSAA_E_POSITION_CALC_FAILURE | The position in the parmBuffer of the requested values was miscalculated |
| u | PGSAA_E_THREED_READ_ERROR | Function failure to read parameter values from buffer |
| l | PGSAA_E_EXTRACTORESULTSERROR | Failure to transfer selected values from parmBuffer to results |
| ul | PGSAA_E_OUTOFRANGE | Input values out of data set range |

## EXAMPLE:

C:

```
PGSt_SMF_status retStatus;

char parms[PGSs_AA_MAXNOCACHES][100] = {
                        "nmcRucSigPres","nmcRucSigPot"};
      PGSt_integer xStart = 30;
      PGSt_integer yStart = 20;
      PGSt_integer zStart = 2;
      PGSt_integer xDim = 6;
      PGSt_integer yDim = 4;
      PGSt_integer zDim = 2;
      PGSt_integer fileId = 10972;  /* contains interleaved
                                       parms */
```

```
float results[2][4][6][2];  /* height,lat,long,parm */

PGSt_integer      nParms = 2

PGSt_SMF_status = PGS_AA_3DRead (iparms, nParms, xStart,
                                 yStart, zStart, xDim, yDim,
                                 zDim, fileId, 1, 2
                                 results);
```

FORTRAN:

```
implicit none

include     "PGS_AA_10.f"
include     "PGS_AA.f"
include     "PGS_SMF.f"

integer           pgs_aa_3dread
character*99      parms(PGSd_AA_MAXNOCACHES)
integer           nParms
integer           xStart
integer           yStart
integer           zStart
integer           xDim
integer           yDim
integer           zDim
integer           fileId
integer           version
integer           operation

integer           result(2,50,20,2)
parms(1)= "nmcRucSigPot"
parms(2)= "nmcRucSigPres"
nParms=2
yStart=102
xStart=205
zStart=2
yDim=20
xDim=50
zDim=2
fileId=10972
operation=2
version = 1

call pgs_aa_3dread( iparms, nparms, xStart, yStart, zStart,
                    xDim, yDim, zDim, fileId, version,
                    operation, results )
```

**NOTES:** For further details of the background to these tools and the available data sets, support files and the means by which new data sets can be introduced, see the Toolkit Primer Ancillary Data section (info on how to access this document can be found in the preface of the Users Guide) or Appendix D in this document. These also include details of the operations that can be set by the user and the autoOperations associated with particular data sets.

The FORTRAN result argument returned is not specified since it depends on the data set used; e.g., it could be real or integer.

The upper limit of the range of input variables is data set specific. The parms input variable is a parameter and data set specific set of strings. The results buffer is a memory buffer holding whatever data is extracted form the data set requested by the user. It can hold data of 4 types (long, short, float, double).

It is critical that the results buffer be declared to be of the same type as that found in the first element of the support file (or PGSt_integer for short /long when working in FORTRAN) and be dimensioned to exactly contain the requested dimensions.

**REQUIREMENTS:** PGSTK–0931, PGSTK–1360, PGSTK–1362

### 6.3.3 Celestial Body Position Tools

The tools included in this section provide the user with information about the locations of celestial bodies (sun, moon, major planets and bright stars). The vector from the Earth or the spacecraft can be computed and the presence of a body in the instrument field of view can be detected.

#### 6.3.3.1 Celestial Body Position Tool Notes

The following notes apply to several of the Celestial Body Position Tools.

**TIME RANGE OF CELESTIAL BODY EPHEMERIS**

The EOSDIS version of the JPL DE200 ephemeris which is used for the celestial body positions is valid from Dec 14, 1949 through Jan 1, 2021. The user's calling times are internally translated to TDT (dynamical time, similar to the old "ephemeris time") before being used to access the epehemeris itself. This translation depends on leap seconds information. If the leap seconds file is not up to date the error message "PGSTD_NO_LEAP_SECS" is returned but processing continues. Since leap seconds are normally available only six months in advance, results for far future simulations cannot be guaranteed. On the other hand, as time passes, with the leap seconds file properly updated by automatic Toolkit procedures, the positions calculated at any given time, for past times, for the present date, or a few months in advance will be reliable.

**TIME OFFSETS:**

These functions accept an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'i' of the input data will be processed at the time: (initial time) + (offset[i]).

Examples:

> if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then input[0] will be processed at time 432000.0 and return output[0]
>
> if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then input[0] will be processed at time 432000.0 + offsets[0] and
> return output[0]
>
> if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI: 432000.0),
> then each input[i] will be processed at time 432000.0 + offsets[i] and
> the result will be output[i], where i is on the interval [0,N)

**ERROR HANDLING:**

These functions process data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of these functions will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of these functions will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of these functions will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status level of 'W' does not necessarily mean that some of the data could not be processed. The 'W' level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will have a status level of 'W' to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS above) that processing was not successful at some of the requested times. Therefore in the case of any return status of level 'W', the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

**EPHEMERIS AND ATTITUDE DATA QUALITY CONTROL:**

Some of the Celestial Body Positioning tools access spacecraft ephemeris and/or attitude data in order to effect their respective transformations. In these cases users may define "masks" for the two data quality flags (ephemeris and attitude) associated with spacecraft ephemeris data. The quality flags are (currently) four byte entities (may be 8 bytes on the cray but only the first four bytes will be considered) that are interpreted bit by bit for meaning (see Section L.3 Quality Flags). Currently the only "fatal" bit (i.e. indicating meaningless data) that will be set prior to access by the Toolkit is bit 16 (where the least significant bit is bit 0). Additionally, the Toolkit will set bit 12 of the quality flag returned for a given user input time if NO data is found for that input time. Note that this usage is different from most of the other bits which indicate the state of some existing data point. By default the Toolkit will set the mask for each of the quality flags to include bit 16 (fatally flawed data) and bit 12 (no data). This means that any data points returned from the tool PGS_EPH_EphemAttit() with an associated quality flag that has either bit 12 or bit 16 set will be rejected by any TOOLKIT function that makes a call to PGS_EPH_EphemAttit() (e.g. these CBP tools) (note that masking is not applied in the tool

PGS_EPH_EphemAttit() itself since users calling this tool directly can examine the quality flags themselves and make their own determination as to which data points to use or reject).

Users may use the Process Control File (PCF) to define their own masks which the Toolkit will then use instead of the defaults mentioned above. The user defined mask should set any bit which the user considers fatal for their purpose (e.g. red limit exceeded). WARNING: if the user defined mask does not have bit 16 set, the Toolkit will pass through data the associated quality flag of which has bit 16 set. The toolkit will not, however, process any data points if the associated quality flag has bit 12 set (i.e. no data exists) whether or not the user mask has bit 12 explicitly set.

Below are the PCF entries which control the value of these masks:

```
# ----------------------------------------------------------------------
# The following parameter is a "mask" for the ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
# with the quality flag should be REJECTED/IGNORED).
# ----------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ----------------------------------------------------------------------
# The following parameter is a "mask" for the attitude data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the attitude data quality flag that
# should be considered fatal (i.e. the attitude data associated
# with the quality flag should be REJECTED/IGNORED).
# ----------------------------------------------------------------------
10508|attitude data quality flag mask|65536
```

Note that in the examples above, the value 65536 is the unsigned integer equivalant of a 32 bit binary counter with bits 12 and 16 set. See section 6.2.3 (Process Control Tools) and (Appendix C Process Control Files) for a detailed explanation of the use of the Process Control File.

**REFERENCES:**

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project", Document 445-TP-002-002, May 1995, by P. Noerdlinger.

# Compute Earth to Celestial Body ECI Vector

**NAME:** **PGS_CBP_Earth_CB_Vector( )**

**SYNOPSIS:**

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_Earth_CB_Vector(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer        cbId,
        PGSt_double         cbVectors[][3])
```

FORTRAN:

```
include'PGS_CBP.f'
include'PGS_TD.f'
include'PGS_SMF.f'
include'PGS_CBP_6.f'
include'PGS_TD_3.f'

integer function
pgs_cbp_earth_cb_vector(numvalues,asciiutc,offsets,cbid,cbvectors)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             cbid
        double precision    cbvectors(3,*)
```

**DESCRIPTION:** This function computes the Earth–Centered Inertial (ECI J2000) frame vector from the Earth to the selected bodies of Solar System.

**INPUTS**

***Table 6-165. PGS_CBP_Earth_CB_Vector Inputs***

| NAME | DESCRIPTION | UNITS | MIN | MAX |
|---|---|---|---|---|
| asciiUTC | UTC time in CCSDS ASCII Time Code A OR B format | time | 1961–01–01 | see NOTES |
| offsets | array of offsets of each input UTC time | seconds | see NOTES | see NOTES |
| cbId | identifier of celestialbody (see list below) | N/A | 1 | 13 |
| numValues | number of required data points<br>0—only asciiUTC in used<br>any—any time events are used | N/A | 0 | any |

THE DESIGNATION OF THE ASTRONOMICAL BODIES BY CELESTIAL BODY IDENTIFIER ( cbId ) IS:

cbId =

| | |
|---|---|
| 1 = MERCURY | 8 = NEPTUNE |
| 2 = VENUS | 9 = PLUTO |
| 3 = EARTH | 10 = MOON |
| 4 = MARS | 11 = SUN |
| 5 = JUPITER | 12 = SOLAR–SYSTEM BARYCENTER |
| 6 = SATURN | 13 = EARTH–MOON BARYCENTER |
| 7 = URANUS | |

**OUTPUTS:**

***Table 6-166. PGS_CBP_Earth_CB_Vector Outputs***

| NAME | DESCRIPTION | UNITS | MIN | MAX |
|---|---|---|---|---|
| cbVectors[][3] | ECI unit vectors from Earth to celestial body first subscript for each time event specified second subscript gives position vector | meter | see NOTES | see NOTES |

**RETURNS:**

***Table 6-167. PGS_CBP_Earth_CB_Vector Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful completion |
| PGSCBP_W_EARTH_CB_ID | Earth cbId is specified |
| PGSCBP_E_INVALID_CB_ID | Invalid celestial body identifier |
| PGSTD_E_BAD_INITIAL_TIME | Initial input time can not be deciphered |
| PGSCBP_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSCBP_E_UNABLE_TO_OPEN_FILE | Ephemeris file can not be opened |
| PGSCBP_E_TIME_OUT_OF_RANGE | Initial time is outside the ephemeris bounds |
| PGSTD_E_NO_LEAP_SECS | No leap second correction available |
| PGSCBP_W_BAD_CB_VECTOR | One or more errors in CB vectors |
| PGS_E_TOOLKIT | For unknown errors |

**EXAMPLES:**

C:

```
#define ARRAY_SIZE      3

PGSt_SMF_status   returnStatus;
PGSt_integer      cbId = 10;
PGSt_integer      numValues;
char              asciiUTC[28] = "2002-07-
                  27T11:04:57.987654Z";
PGSt_double       offsets[ARRAY_SIZE] = {3600.0, 7200.0,
                  10800.0};
PGSt_double       cbVectors[ARRAY_SIZE][3];

char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

numValues = ARRAY_SIZE;

returnStatus = PGS_CBP_Earth_CB_Vector(numValues, asciiUTC,
                                       offsets, cbId,
                                       cbVectors)

if (returnStatus != PGS_S_SUCCESS)
{
    PGS_SMF_GetMsg(&returnStatus, err, msg);
    printf ("ERROR: %s\n", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_earth_cb_vector
integer           returnstatus
integer           cbid
integer           numvalues

double precision  offsets(3)
double precision  cbvectors(3,3)

character*27      asciiutc
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = '2002-07-27T11:04:57.987654Z'
cbid = 10
numvalues = 3

returnstatus = pgs_cbp_earth_cb_vector(numvalues, asciiutc,
                                       offsets, cbid,
                                       cbvectors)
```

```
if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err, msg
endif
```

**NOTES:** See Section 6.3.3.1 Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

**REQUIREMENTS:** PGSTK–0800

# Compute Satellite to Celestial Body Vector in Spacecraft Reference Frame

**NAME:** **PGS_CBP_Sat_CB_Vector( )**

**SYNOPSIS:**

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_Sat_CB_Vector(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_integer        cbId,
        PGSt_double         cbVectors[][3])
```

FORTRAN:

```
include 'PGS_CBP.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'
include 'PGS_CBP_6.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'

integer function
pgs_cbp_sat_cbvectors(spacecrafttag,numvalues,asciiutc,offsets,cbid,
                      cbvectors)

        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             cbid
        double precision    cbvectors(3,*)
```

**DESCRIPTION:** This function computes the vector in the spacecraft reference frame from the spacecraft to the sun, moon, or planets at a given time or range of times.

**INPUTS:**

*Table 6-168. PGS_CBP_Sat_CB_Vector Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of required data points:<br>0—only asciiUTC is used<br>any—any time events are used | N/A | 0 | any |
| asciiUTC [28] | UTC time in CCSDS ASCII Time code A or B format | time | 1961–01–01 | see NOTES |
| offsets[] | array of time offsets from asciiUTC in seconds | seconds | see NOTES | see NOTES |
| cbId | identifier of celestial bodies (see list below) | N/A | N/A | N/A |

THE DESIGNATION OF THE ASTRONOMICAL BODIES BY CELESTIAL BODY IDENTIFIER ( cbId ) IS:

cbId =

1 = MERCURY
2 = VENUS
3 = EARTH
4 = MARS
5 = JUPITER
6 = SATURN
7 = URANUS
8 = NEPTUNE
9 = PLUTO
10 = MOON
11 = SUN
12 = SOLAR–SYSTEM BARYCENTER
13 = EARTH–MOON BARYCENTER

**OUTPUTS:**

*Table 6-169. PGS_CBP_Sat_CB_Vector Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| cbVectors[][3] | vectors in spacecraft reference frame from satellite to the celestial body for each time event | meter | see NOTES | see NOTES |

**RETURNS:**

*Table 6-170. PGS_CBP_Sat_CB_Vector Returns (1 of 2)*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_W_BELOW_SURFACE | Output vector from ECItoSC below surface |
| PGSCBP_W_BAD_CB_VECTOR | One or more bad vectors for requested times |
| PGSCBP_E_BAD_ARRAY_SIZE | numvalues is less than 0 |
| PGSCBP_E_INVALID_CB_ID | Invalid celestial body identifier |
| PGSMEM_E_NO_MEMORY | Not enough memory for tmpVectors |

*Table 6-170. PGS_CBP_Sat_CB_Vector Returns (2 of 2)*

| Return | Description |
|---|---|
| PGSCBP_E_UNABLE_TO_OPEN_FILE | Unable to open planetary data file |
| PGSTD_E_BAD_INITIAL_TIME | Initial time is incorrect |
| PGSCBP_E_TIME_OUT_OF_RANGE | Initial time is outside the ephemeris bounds |
| PGSTD_E_SC_TAG_UNKNOWN | Invalid spacecraft tag |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |
| PGS_E_TOOLKIT | Toolkit error |

**EXAMPLES**:

C:

```
#define ARRAY_SIZE      3

PGSt_SMF_status         returnStatus;

PGSt_integer            numValues = ARRAY_SIZE;

PGSt_double             cbVectors[ARRAY_SIZE][3];
PGSt_double             offsets[ARRAY_SIZE] = {3600.0,
                                   7200.0, 10800.0};

char                    asciiUTC[28] = "2002-07-
                              27T11:04:57.987654Z";
char                    err[PGS_SMF_MAX_MNEMONIC_SIZE];
char                    msg[PGS_SMF_MAX_MSG_SIZE];

returnStatus = PGS_CBP_Sat_CB_Vector(PGSd_EOS_AM, numValues,
                                       asciiUTC, offsets,
                                 PGSd_MOON, cbVectors);

if (returnStatus != PGS_S_SUCCESS)
{
    PGS_SMF_GetMsg(&returnStatus, err, msg);
    printf ("ERROR: %s\n", msg);
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_sat_cb_vector
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
integer           cbid
double precision  cbvectors(3,3)
```

```
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = "2002-07-27T11:04:57.987654Z"
cbid = 10
numvalues = 3

returnstatus = pgs_cbp_sat_cb_vector(pgsd_eos_am, numvalues,
>                                    asciiutc, offsets,
>                                    pgsd_moon, cbvector)

if (returnstatus .ne. pgs_s_success) then
     pgs_smf_getmsg(returnstatus, err, msg)
     write(*,*) err, msg
endif
```

**NOTES:** See Section 6.3.3.1. Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS:** PGSTK–0680, PGSTK–0810

# Get Solar Time and Coordinates

**NAME:** **PGS_CBP_SolarTimeCoords( )**

**SYNOPSIS:**

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CBP_SolarTimeCoords(
        char            asciiUTC[28],
        PGSt_double     longitude,
        PGSt_double     *meanSolTimG,
        PGSt_double     *meanSolTimL,
        PGSt_double     *apparSolTimL,
        PGSt_double     *solRA,
        PGSt_double     *solDec)
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD_3.f'

integer function pgs_cbp_solartimecoords(asciiutc, longitude,
                                         meansoltimg, meansoltiml,
                                         apparsoltiml, solra, soldec)
        character*27        asciiutc
        double precision    longitude
        double precision    meansoltimg
        double precision    meansoltiml
        double precision    apparsoltiml
        double precision    solra
        double precision    soldec
```

**DESCRIPTION** This tool performs a low accuracy rapid calculation of solar time and coordinates. The accuracy of the equations here is expected to be about 0.5 minutes of time and 0.04 degrees for the coordinates of the sun.

**INPUTS:**

*Table 6-171. PGS_CBP_SolarTimeCoords Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| asciiUTC | Coordinated Universal Time in CCSDS ASCII Time Code A or B format | N/A | See NOTES | See NOTES |
| longitude | longitude of observer (positive is East) Not required for solar coordinates; should be set to 0 in that case | radians | -pi | pi |

**OUTPUTS:**

*Table 6-172. PGS_CBP_SolarTimeCoords Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| meanSolTimG | Greenwich Mean Solar Time as seconds from midnight | seconds | 0 | 86400 |
| meanSolTimL | Local Mean Solar Time as seconds from midnight | seconds | 0 | 86400 |
| apparSolTimL | Local Apparent Solar Time as seconds from midnight | seconds | 0 | 86400 |
| solRA | Right Ascension of the Mean sun | radians | 0 | 2*pi |
| solDec | Declination of the Mean sun | radians | -pi | pi |

**RETURNS:**

*Table 6-173. PGS_CBP_SolarTimeCoords Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful execution |
| PGSTD_M_LEAP_SEC_IGNORED | Input leap second has been ignored |
| PGSTD_E_TIME_FORMAT_ERROR | Error in format of input ASCII UTC time |
| PGSTD_E_TIME_VALUE_ERROR | Error in value of input ASCII UTC time |
| PGS_E_TOOLKIT | Something unexpected happened, execution aborted |

**EXAMPLES:**

C:

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       longitude;
PGSt_double       meanSolTimG;
PGSt_double       meanSolTimL;
```

```
PGSt_double       solRA;
PGSt_double       solDec;

strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CBP_SolarTimeCoords(asciiUTC,longitude,
                                       &meanSolTimG,
                                       &meanSolTimL,
                                       &apparSolTimL,
                                       &solRA,&solDec)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
printf("\n longitude: %lf",longitude);

printf("Greenwich Mean Solar Time:%lf  Local Mean Solar
        Time:%lf", meanSolTimG,meanSolTimL);
printf("\n Local Apparent Solar Time:%lf  Solar Right
        Asc/Dec:%lf/%lf", apparSolTimL,solRA,solDec);
```

FORTRAN:

```
implicit none

integer           pgs_cbp_solartimecoords
character*27      asciiutc
double precision  longitude
double precision  meansoltimg
double precision  meansoltiml
double precision  apparsoltiml
double precision  solra
double precision  soldec
integer           returnstatus

asciiutc = '1991-01-01T11:29:30'
longitude = 1.0

returnstatus = pgs_cbp_solartimecoords(asciiutc,longitude,
                                       meansoltimg,
                                       meansoltiml,
                                       apparsoltiml,solra,
                                       soldec)

if(returnstatus .ne. pgs_s_success) go to 90
write(6,*) asciiutc,longitude
write(6,*)meansoltimg,meansoltiml,apparsoltiml,solra,soldec
```

```
90 write(6,99)returnstatus

99 format('ERROR:',I50)
```

**NOTES:** The equations used in this function are referenced on page C24 of the 1994 Astronomical Almanac. They are low precision formulas that give the apparent coordinates of the sun to a precision of 0.01 degrees and the equation of time to a precision of 0.5 minutes between the years 1950 and 2050. Less accuracy is expected for dates before 1950 and after 2050.

More accurate solar time determination requires improved solar coordinates and the value of UT1–UTC. These items are accessible through other SDP tools.

In particular, the Solar ephemeris yields accurate solar coordinates and the function PGS_TD_gmst() gives Greenwich Mean Sidereal Time. These can be combined to obtain more accurate Mean Solar Time. The difference UT1–UTC is determined within the coordinate system conversion (CSC) group of functions, in the transformations between Earth Centered Rotating (ECR) and Earth Centered Inertial (ECI).

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

**REQUIREMENTS:** PGSTK–0760

# Celestial Body in Field–of–View Indicator

**NAME:** **PGS_CBP_body_inFOV( )**

**SYNOPSIS:**

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_CBP.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CBP_body_inFOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[][3],
        PGSt_double         *perimFOV_vectors,
        PGSt_tag            cbID,
        PGSt_boolean        inFOVflag[],
        PGSt_double         cb_vector[][3],
        PGSt_double         cb_SCvector[][3])
```

FORTRAN:

```
include 'PGS_TD_3.f'
include 'PGS_CSC_4.f'
include 'PGS_CBP_4.f'
include 'PGS_EPH_4.f'
include 'PGS_MEM_4.f'
include 'PGS_SMF.f'

integer function pgs_cbp_body_infov(numvalues,asciiutc,offsets,
                                    spacecrafttag,numfovperimvec,infovvector,
                                    perimfov_vectors,cbid,infovflag,cb_vector,
                                    cb_scvector)

integer             numvalues
character*27        asciiutc
double precision    offsets(*)
integer             spacecrafttag
integer             numfovperimvec
double precision    infovvector(*)
double precision    perimfov_vectors(3,numfovperimvec,*)
```

| | |
|---|---|
| integer | cbid |
| integer | infovflag(*) |
| double precision | cb_vector(3,*) |
| double precision | cb_scvector(3,*) |

**DESCRIPTION:** Given a celestial body (CB) identifier (as in the CBP tools) and a field of view (FOV) description, tool returns a flag or flags indicating if the CB is in the FOV, as well as the coordinates of the CB in SC coordinates. Alternatively, the user can specify CB identifier 999 or PGSd_STAR and supply the ECI vector to the body.

**INPUTS:**

***Table 6-174. PGS_CBP_body_inFOV Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of time gridpoints | N/A | 1 | any |
| asciiUTC | UTC start time | N/A | 1979–06–30T00:00:01 | 2008–01–01T 12:00:00 |
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numFOVperimVec | number of vectors defining FOV perimeter | N/A | 3 | any |
| inFOVvector | vector in FOV, in SC coordinates | N/A | N/A | N/A |
| perimFOV_vectors | vectors in SC coords defining FOV's; MUST be sequential around FOV; middle dimension must be exactly the same value as numFOVperimVec because of the way the array dimensioning works in the function. | N/A | N/A | N/A |
| cbId | celestial body ID (Earth not included—see PGS_CSC_Earthpt_FOV) | N/A | 1 | 13 |
| cb_vector | ECI vectors of CB (this is an input only when cbId = 999, meaning user input of ECI vector for CB—see notes) | Arbitrary | see PGS_CBP_Earth_CB _Vector | |

**OUTPUTS:**

*Table 6-175. PGS_CBP_body_inFOV Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| inFOVflag | PGS_TRUE if CB is in FOV—see notes | N/A | N/A | N/A |
| cb_SCvector | vector of CB in SC coords notes | meters | see PGS_CBP_body_inFOV() notes | |

**RETURNS:**

*Table 6-176. PGS_CBP_body_inFOV Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCBP_E_TIME_OUT_OF_RANGE | Initial time is outside the ephemeris bounds |
| PGSTD_E_BAD_INITIAL_TIME | Initial time is incorrect |
| PGSCBP_W_BAD_CB_VECTOR | One or more bad vectors for requested times |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSCSC_W_DATA_FILE_MISSING | The data file earthfigure.dat is missing |
| PGSCBP_E_UNABLE_TO_OPEN_FILE | Unable to open file |
| PGSCBP_E_INVALID_CB_ID | Invalid celestial body identifier |
| PGSCBP_W_EARTH_CB_ID | The tool PGS_CSC_Earthpt_FOV() must be used to check for Earth points in the FOV |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSCSC_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |
| PGSEPH_E_NO_DATA_REQUESTED | Both orb and att flags are set to false |
| PGSCSC_E_INVALID_FOV_DATA | FOV perimeter vectors are invalid |
| PGSCSC_E_FOV_TOO_LARGE | FOV specification outside algorithmic limits |
| PGS_E_TOOLKIT | Something unexpected happened |

**EXAMPLES**:

C:

```
#define          ARRAY_SIZE   3
#define          PERIMVEC_SIZE 4
PGSt_SMF_status  returnStatus;
char             asciiUTC[28];
PGSt_double      offsets[ARRAY_SIZE] =
                       {3600.0,7200.0,10800.0};
PGSt_integer     numValues;
PGSt_integer     numFOVperimVec;
```

```
PGSt_double       inFOVvector[ARRAY_SIZE][3] =
                        { {0.0,0.0,100.0},
                          {0.0,0.0,200.0},
                          {0.0,0.0,300.0}
                          };
PGSt_double
      perimFOV_vectors[ARRAY_SIZE][PERIMVEC_SIZE][3]=
                      { {100.0,100.0,100.0},
                        {-100.0,100.0,100.0},
                        {-100.0,-100.0,100.0},
                        {100.0,-100.0,100.0},
                        {200.0,200.0,200.0},
                        {-200.0,200.0,200.0},
                        {-200.0,-200.0,200.0},
                        {200.0,-200.0,200.0},
                        {300.0,200.0,200.0},
                        {-200.0,300.0,200.0},
                        {-200.0,-300.0,300.0},
                        {300.0,-200.0,200.0},
                      };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       cb_SCvector[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");

returnStatus = PGS_CBP_body_inFOV(numValues,asciiUTC,
                                  offsets,PGSd_TRMM,
                                  numFOVperimVec,
                                  inFOVvector,
                                  perimFOV_vectors,
                                  PGSD_MOON,
                                  inFOVflag,NULL,
                                  cb_SCvector);

if(returnStatus != PGS_S_SUCCESS)|
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_cbp_body_infov
integer           returnstatus
integer           spacecrafttag
```

```
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
integer           spacecrafttag
integer           numfovperimvec
double precision  infovvector(3,3)
double precision  perimfov_vectors(3,4,3)
integer           cbid
integer           infovflag(3)
double precision  cb_vector(3,3)
double precision  cb_scvector(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

infovvector(1,1) = 0.0
infovvector(1,2) = 0.0
infovvector(1,3) = 0.0

infovvector(2,1) = 0.0
infovvector(2,2) = 0.0
infovvector(2,3) = 0.0

infovvector(3,1) = 0.0
infovvector(3,2) = 0.0
infovvector(3,3) = 0.0

perimfov_vectors(1,1,1) = 100.0
perimfov_vectors(2,1,1) = 100.0
perimfov_vectors(3,1,1) = 100.0

perimfov_vectors(1,2,1) = -100.0
perimfov_vectors(2,2,1) = 100.0
perimfov_vectors(3,2,1) = 100.0

perimfov_vectors(1,3,1) = -100.0
perimfov_vectors(2,3,1) = -100.0
perimfov_vectors(3,3,1) = 100.0

perimfov_vectors(1,4,1) = 100.0
perimfov_vectors(2,4,1) = -100.0
perimfov_vectors(3,4,1) = 100.0

perimfov_vectors(1,1,2) = 200.0
perimfov_vectors(2,1,2) = 200.0
perimfov_vectors(3,1,2) = 200.0
```

```
perimfov_vectors(1,2,2) = -200.0
perimfov_vectors(2,2,2) = 200.0
perimfov_vectors(3,2,2) = 200.0

perimfov_vectors(1,3,2) = -200.0
perimfov_vectors(2,3,2) = -200.0
perimfov_vectors(3,3,2) = 200.0

perimfov_vectors(1,4,2) = 200.0
perimfov_vectors(2,4,2) = -200.0
perimfov_vectors(3,4,2) = 200.0

perimfov_vectors(1,1,3) = 300.0
perimfov_vectors(2,1,3) = 300.0
perimfov_vectors(3,1,3) = 300.0

perimfov_vectors(1,2,3) = -300.0
perimfov_vectors(2,2,3) = 300.0
perimfov_vectors(3,2,3) = 300.0

perimfov_vectors(1,3,3) = -300.0
perimfov_vectors(2,3,3) = -300.0
perimfov_vectors(3,3,3) = 300.0

perimfov_vectors(1,4,3) = 300.0
perimfov_vectors(2,4,3) = -300.0
perimfov_vectors(3,4,3) = 300.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4

returnstatus = pgs_cbp_body_infov(numvalues,asciiutc,
                                  offsets,PGSd_TRMM,
                                  numfovperimvec,
                                  infovvector,
                                  perimfov_vectors,moon,
                                  infovflag,null,
                                  cb_scvector);

if (returnstatus .ne. pgs_s_success) then
     pgs_smf_getmsg(returnstatus, err, msg)
     write(*,*) err, msg
endif
```

**NOTES:** The FOV is always specified in SC coordinates; for an instrument fixed to the SC, use the same FOV description always; for scanning instruments, user should provide the description appropriate to the scan instant.

numFOVperim must be at least 3. The tool determines if any part of the CB requested lies within the perimeter defined by the vectors perimFOV_vectors[][][3]. The first index in C (last in FORTRAN) is the time offset index, and the second MUST be sequential around the FOV

perimeter. The vector inFOVvector[][3] MUST lie within the FOV. It need not be central, but there will be loss of efficiency if not. The last index in C (first in FORTRAN) on these vectors is for X,Y and Z components in SC coordinates. It is necessary for the user to supply a vector within the FOV for the reason that on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside FOV, which outside.

The vectors "perimFOV_vectors[][][3]" defining the FOV perimeter can be in clock- or counter-clockwise sequence .

The tool may be used on the Sun, Moon, and planets other than the Earth, in which case the cbID must be selected from the standard set (see the tool PGS_CBP_Earth_CB_Vector()). The tool may also be used on another object (such as a star), in which case cbID should be set = 999 and the ECI J2000 coordinates of the star must be supplied in cb_vector[]. The Sun, Moon and planets have finite radii, as specified in the Table below; CB's with cdID = 999 (PGSd_STAR) are assumed to be of negligible radius.

Note on Finite Size of CB: Since a primary use of this tool will be to determine if the Sun, Moon, or a planet intrudes into the FOV, it is important to allow for the finite size of the object. For this purpose, the Moon and Planets are replaced with spheres of the following radii, which are projected on the celestial sphere:

***Table 6-177. Physical Radii for CB in FOV Tool***

| CB | Radius (km) | Explanation |
|---|---|---|
| Sun | 7 e 5 | |
| Moon | 1739 | allows for topography |
| Mercury | 2440 | |
| Venus | 6055 | |
| Earth | n/a | use tool PGS_CSC_Earthpt_FOV() |
| Mars | 3397 | ignore satellites |
| Jupiter | 1890 e 3 | include Galilean satellites |
| Saturn | 1225 e 3 | include rings, satellites to Titan |
| Uranus | 25600 | planet only |
| Neptune | 24800 | planet only |
| Pluto | 19600 | planet and Charon |
| 999 (STAR) | 0.0 | a star, or user–defined point |

In general, we have included satellites down to the 10th magnitude.

In the case that the celestial body position is invalid for a particular time, then the corresponding cb_SCvector will be set to PGSd_GEO_ERROR_VALUE.

If the CB disk overlaps the FOV only behind the Earth's equatorial bulge and the overlap is barely hidden by it, and the FOV has a sharp corner protruding past the Earth limb it is possible in rare cases that a false positive answer will issue.

See Section 6.3.3.1 Celestial Body Position Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS:** PGSTK–0780

### 6.3.4 Coordinate System Conversion Tools

#### 6.3.4.1 Introduction

The ECI system is J2000. Thus in Fig. 6-2 the Z axis is along the Earth's rotation axis at the epoch of J2000. The ECR system is Earth fixed, i.e. rotating with the Earth. Since its definition includes the effect of polar motion, then in Fig. 6-1 the Z axis is actually along geographic North, which differs very slightly and variably (~ 3 to 15 meters) from the rotational North axis.

#### 6.3.4.2 Unit Vectors for Input

In some functions, a unit vector or a set of unit vectors is required on input. In these cases, the vectors are generally renormalized internally again, anyway, to prevent obscure errors. Thus, generally, any vector defining the correct direction can be supplied; it need not be normalized. An exception is in the transformations between ECI and Spacecraft Coordinates, PGS_CSC_ECItoSC( ) and PGS_CSC_SCtoECI( ). In these functions, the behavior is different for unit vector input and for input vectors in meters. For these two functions, any input vector whose length is between 0.99999 and 1.000001 is assumed to be a unit vector, while any other vector is assumed to be measured in meters.

#### 6.3.4.3 Other Specialized Vectors and Terminology

The vector along a line of sight from the spacecraft may be referred to as a "pixel vector" or "look vector". It could be the boresight of an instrument or it could locate a point in a finite field of view. The "look point" is the intersection of such a vector with the Earth ellipsoid. Vectors designated by the name of a celestial body, such as the "Sun vector" are assumed to point from Earth center or spacecraft center to the celestial body, depending on context. They are in meters unless described as unit vectors. "Latitude" always means geodetic latitude, and the zenith vector at Earth surface is always taken as the normal to the Earth ellipsoid. The "slant range" is from the instrument boresight or spacecraft center to the look point (depending on the accuracy flag). The "field of view" is always defined in spacecraft coordinates. The "subsatellite point" is at the foot of a normal dropped from the spacecraft to the Earth ellipsoid, and its velocity what would be measured by terrestrial instruments on the ellipsoid.

#### 6.3.4.4 Altitudes; Altitude Warnings

In almost all the tools, such as transformations between ECR and Geodetic coordinates, in PGS_CSC_GetFOV_Pixel( ), and PGS_CSC_SubSatPoint( ) the altitude is defined in meters off the Earth ellipsoid, as specified by the user through an Earth ellipsoid tag. If an invalid tag is used, then the WGS84 ellipsoid is used. The altitude for PGS_CSC_ZenithAzimuth( ) is the exception; it must be defined in meters off the *geoid*, because it is used to calculate air density to correct the zenith angle for refraction when that correction is requested. The altitude is ignored otherwise in PGS_CSC_ZenithAzimuth( ). All the functions that input or output altitude, or calculate it internally check for reasonableness. When large negative altitudes are input or generated internally, warning messages issue to the log file, and a warning return will be given unless there is a more serious problem. The exact depth used to trigger a warning varies

according to context. For example, in PGS_CSC_ECItoSC( ), the depth can be as great as 0.02 Earth radii, on the supposition that in an extreme case the user might wish the coordinates of some point that deep in the Earth, while in PGS_CSC_Earthpt_FOV( ) and PGS_CSC_Earthpt_FixedFOV(), the warning is issued if the depth exceeds 50 km. Here, the function is not just a coordinate transformation, but it informs the user if the point can be seen. The 50 km tolerance allows that even if the Earth ellipsoid model is set by the user so large as to include most of the atmosphere, and the Earth point is on the ocean floor, no warning will be returned. For greater depths the point is deemed not to be visible and the answer always PGS_FALSE. The maximum altitude of 100 km is set to include noctilucent clouds. Higher altitudes will be processed, with the answer PGS_TRUE or PGS_FALSE according to the geometry, but a warning is issued. This is the only case in which a large positive altitude results in a warning, because of the context that one is talking about an "Earth point."

#### 6.3.4.5 Lines of sight; visibility of points

The various functions do not check for obstruction of the line of sight by part of the spacecraft or clouds, nor for the occultation of one celestial body by another. The tool PGS_CSC_Earthpt_FOV( ) checks for occultation of the specified point by the solid Earth, i.e., on the far side, and PGS_CBP_body_inFOV( ) checks for occultation by the Earth; in those cases a PGS_FALSE answer is reported.

#### 6.3.4.6 Ranges for variables

The minimum and maximum values specified in the tables are often guidelines only and may not be rigidly enforced. For example, a likely range is indicated for any one component of the spacecraft velocity, but, in principle a velocity component could be anything up to escape velocity. When angles such as latitude or longitude are input, they generally are not be checked against the specified ranges. If they are out of range, the results may be unpredictable. The angles are always in radians. An angle inadvertently supplied in degrees will usually lead to a wrong answer rather than an error return. Various mathematical libraries that vendors supply with compilers may also give degraded performance when given angles that are badly out of range.

#### 6.3.4.7 Updating the UT1 and polar motion file

The file $PGSDAT/CSC/utcpole.dat contains information about UT1 and polar motion used by many tools. Since this information changes with time, the file must be periodically updated. The SDP Toolkit contains utilities to perform this update function. If a new leap second is issued, the data in this file will change for dates after that second. Since the IERS can announce a leap second on as little as 90 days notice, the file will contain data for only 83 days after its last update; this allows time for the posting of a new data set by the U.S. N. O. as described below, and for the running of the Toolkit update. Tools that depend on these data, such as transformations between ECR and ECI, and tools that deliver UT1 or sidereal time, will fail and issue an error return if they are provided input times past the end of the file. The Log Status file will indicate the failure with a message including " PGSTD_E_NO_UT1_VALUE".

The shell script update_utcpole.sh, which is found in $PGSBIN, will update the utcpole.dat file to the current date. To maintain a current utcpole.dat, this script should be run every week, but twice a week is recommended for optimum accuracy (<~ 2m). The U.S. Naval Observatory file, on which the update depends, is normally replaced by a current one by noon, Eastern Standard Time, each Tuesday and Thursday. The accuracy is discussed in Section 6.2.7.5.2 . Update_utcpole.sh calls PGS_CSC_UT1_update, a C program that performs most of the actual update work. A Clear Case capable version update_utcpole_CC.sh is provided, as well, with this version of the Toolkit. It must be used from within a Clear Case view belonging to the process owner.

The update is done by collecting the latest information via ftp from United States Naval Observatory in Washington, DC. Their file "finals.data" in the Series 7 directory within server "maia.usno.navy.mil" contains information on UT1-UTC and the x and y pole displacements. The utcpole.dat header contains the date of updating and the file date as listed within ftp for the last "finals.data" used to update it. The function PGS_CSC_UT1_update reformats the new finals.data information and adds it to the utcpole.dat file, overwriting any old information that is superseded. At the DAACs, the process is done automatically by the scheduler. At Science Computing Facilities, for Toolkits through version 5.2.1, drop 4, users will need to have a ".netrc" file in their home directories, as explained in the comments within the scripts. Later releases will not need such a file.

### 6.3.4.8 Coordinate System Conversion Tool Notes

The following notes apply to several of the Coordinate System Conversion Tools.

**TIME OFFSETS:**

These functions accept an ASCII UTC time, an array of time offsets and the number of offsets as input. Each element in the offset array is an offset in seconds relative to the initial input ASCII UTC time.

An error will be returned if the number of offsets specified is less than zero. If the number of offsets specified is actually zero, the offsets array will be ignored. In this case the input ASCII UTC time will be converted to Toolkit internal time (TAI) and this time will be used to process the data. If the number of offsets specified is one (1) or greater, the input ASCII UTC time will be converted to TAI and each element 'i' of the input data will be processed at the time: (initial time) + (offset[i]). It is recommended that users take advantage of the efficiency that can be gained by processing many time values in one run, using offsets. Many of the tools have been designed to run more efficiently when operating in this mode, and in some cases an internal limit ~30 to 50 has been set on error messaging to the log file in this mode, to prevent excessive growth of the log file.

Examples:

> if numValues is 0 and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
> then input[0] will be processed at time 432000.0 and return output[0]

if numValues is 1 and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
then input[0] will be processed at time 432000.0 + offsets[0] and
return output[0]

if numValues is N and asciiUTC is "1993-001T12:00:00" (TAI93: 432000.0),
then each input[i] will be processed at time 432000.0 + offsets[i] and
the result will be output[i], where i is on the interval [0,N)

**ERROR HANDLING:**

These functions process data over an array of times (specified by an input ASCII UTC time and an array of time offsets relative to that time).

If processing at each input time is successful the return status of these functions will be PGS_S_SUCCESS (status level of 'S').

If processing at ALL input times was unsuccessful the status level of the return status of these functions will be 'E'.

If processing at some (but not all) input times was unsuccessful the status level (see SMF) of the return status of this function will be 'W' AND all high precision real number (C: PGSt_double, FORTRAN: DOUBLE PRECISION) output variables that correspond to the times for which processing was NOT successful will be set to the value: PGSd_GEO_ERROR_VALUE. In this case users may (should) loop through the output testing any one of the aforementioned output variables against the value PGSd_GEO_ERROR_VALUE. This indicates that there was an error in processing at the corresponding input time and no useful output data was produced for that time.

Note: A return status with a status level of 'W' does not necessarily mean that some of the data could not be processed. The 'W' level may indicate a general condition that the user may need to be aware of but that did not prohibit processing. For example, if an Earth ellipsoid model is required, but the user supplied value is undefined, the WGS84 model will be used, and processing will continue normally, except that the return status will have a status level of 'W' to alert the user that the default earth model was used and not the one specified by the user. The reporting of such general warnings takes precedence over the generic warning (see RETURNS section of the tool of interest) that processing was not successful at some of the requested times. Therefore in the case of any return status of level 'W', the returned value of a high precision real variable generally should be examined for errors at each time offset, as specified above.

**EPHEMERIS AND ATTITUDE DATA QUALITY CONTROL:**

Many of the Coordinate System Conversion tools access spacecraft ephemeris and/or attitude data in order to effect their respective transformations. In these cases users may define "masks" for the two data quality flags (ephemeris and attitude) associated with spacecraft ephemeris data. The quality flags are (currently) four byte entities (may be 8 bytes on the cray but only the first four bytes will be considered) that are interpreted bit by bit for meaning (see Section L.3 Quality Flags). Currently the only "fatal" bit (i.e. indicating meaningless data) that will be set prior to access by the Toolkit is bit 16 (where the least significant bit is bit 0). Additionally, the Toolkit will set bit 12 of the quality flag returned for a given user input time if NO data is found for that

input time. Note that this usage is different from most of the other bits which indicate the state of some existing data point. By default the Toolkit will set the mask for each of the quality flags to include bit 16 (fatally flawed data) and bit 12 (no data). This means that any data points returned from the tool PGS_EPH_EphemAttit() with an associated quality flag that has either bit 12 or bit 16 set will be rejected by any TOOLKIT function that makes a call to PGS_EPH_EphemAttit() (e.g. these CSC tools) (note that masking is not applied in the tool PGS_EPH_EphemAttit() itself since users calling this tool directly can examine the quality flags themselves and make their own determination as to which data points to use or reject).

Users may use the Process Control File (PCF) to define their own masks which the Toolkit will then use instead of the defaults mentioned above. The user defined mask should set any bit which the user considers fatal for their purpose (e.g. red limit exceeded). WARNING: if the user defined mask does not have bit 16 set, the Toolkit will pass through data the associated quality flag of which has bit 16 set. The toolkit will not, however, process any data points if the associated quality flag has bit 12 set (i.e. no data exists) whether or not the user mask has bit 12 explicitly set.

Below are the PCF entries which control the value of these masks:

```
# ----------------------------------------------------------------------
# The following parameter is a "mask" for the ephemeris data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the ephemeris data quality flag that
# should be considered fatal (i.e. the ephemeris data associated
# with the quality flag should be REJECTED/IGNORED).
# ----------------------------------------------------------------------
10507|ephemeris data quality flag mask|65536
#
# ----------------------------------------------------------------------
# The following parameter is a "mask" for the attitude data quality
# flag.  The value should be specified as an unsigned integer
# specifying those bits of the attitude data quality flag that
# should be considered fatal (i.e. the attitude data associated
# with the quality flag should be REJECTED/IGNORED).
# ----------------------------------------------------------------------
10508|attitude data quality flag mask|65536
```

Note that in the examples above, the value 65536 is the unsigned integer equivalant of a 32 bit binary counter with bits 12 and 16 set. See section 6.2.3 (Process Control Tools) and (Appendix C Process Control Files) for a detailed explanation of the use of the Process Control File.

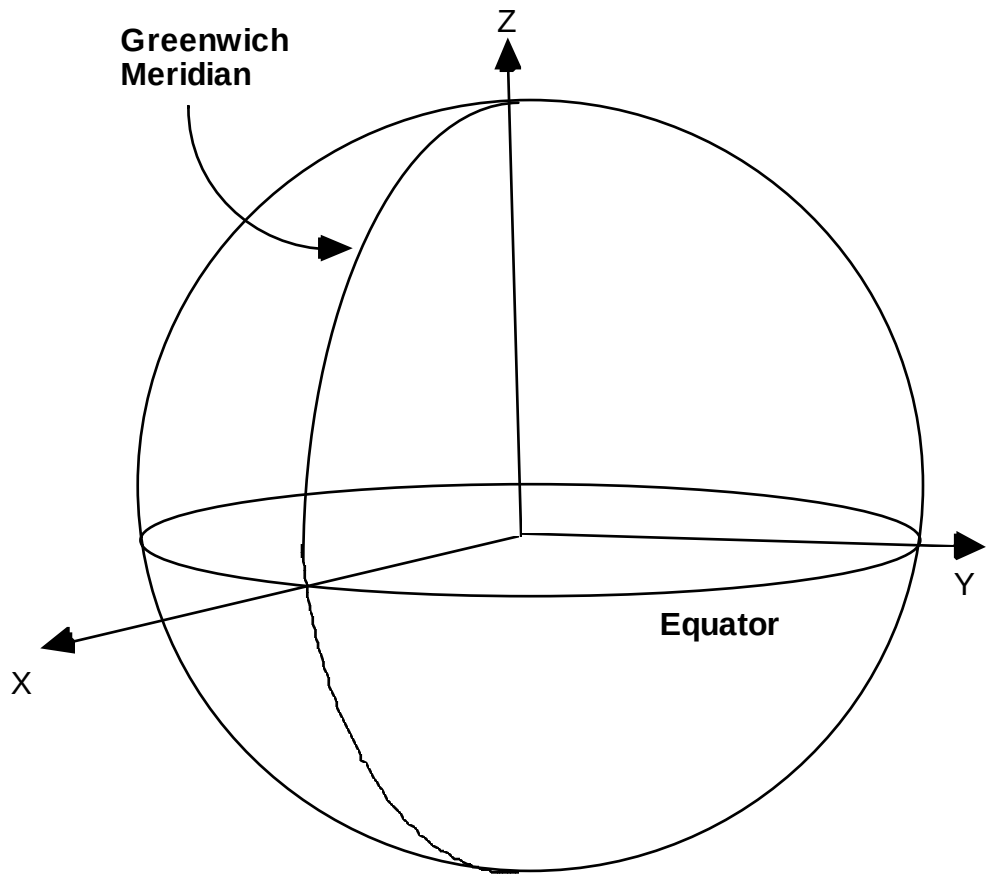
#### 6.3.4.9 Coordinate System Conversion Transformation Tools

These tools convert between various coordinate systems. This will allow calculations to be computed in the most appropriate coordinate system and allow the conversion of results to a common reference frame. Previously these coordinate transformations were contained in one tool entitled PGS_CSC_FrameChange. We have provided separate calls for each transformation,

as one tool proved unwieldy. Also, the user now need not supply extraneous parameters not needed for the desired conversion.

Figures 6–1 through 6–3 show the definitions of the ECR, ECI, and orbital (Orb) reference frames.

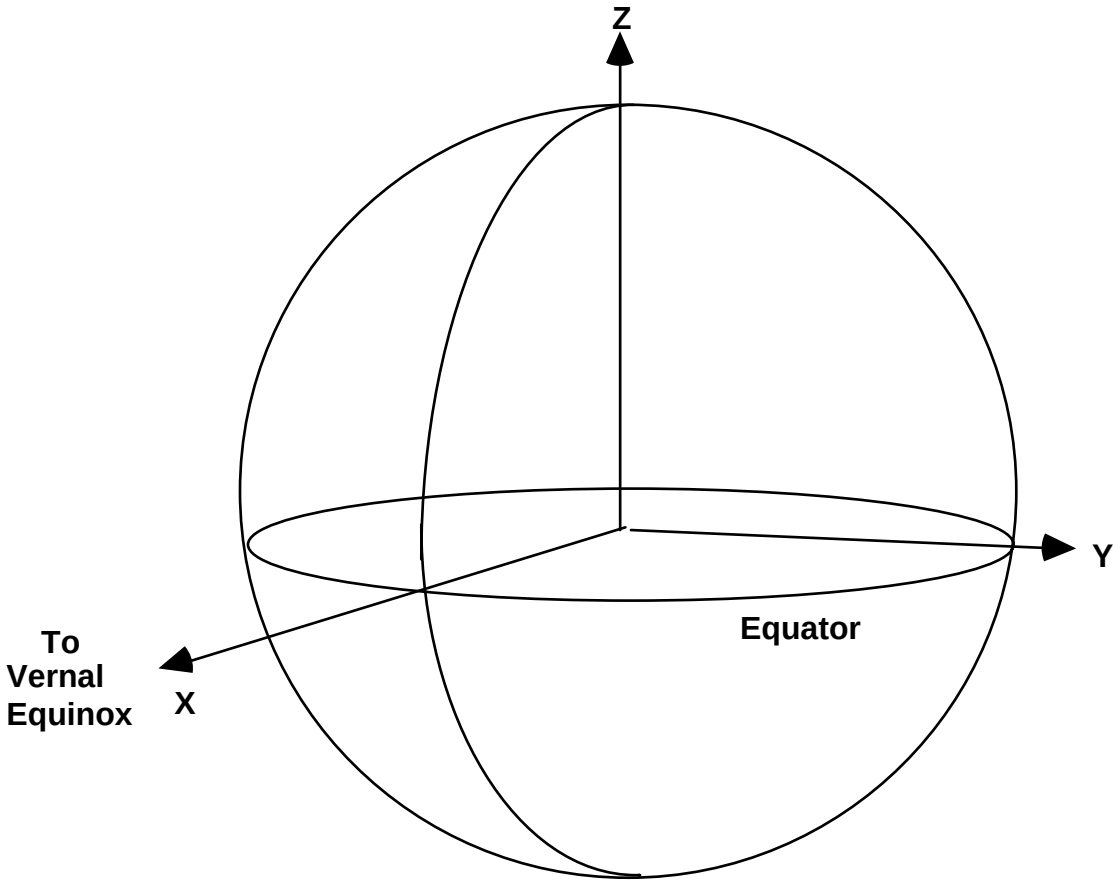
The spacecraft coordinate system coincides with the orbital system when all the Euler angles are zero. Otherwise, it is rotated by the amount indicated by the Euler angles. Thus, a small, positive roll angle indicates that its right side is lowered, and its left side raised. A small positive pitch angle indicates that its nose is raised and thrusters depressed. A small positive zero angle indicates that it is crabbing with its nose to the right of the flight path.




**EARTH-CENTERED ROTATING (ECR) COORDINATES**

Origin: Center of the Earth
Z-Axis: along Earth's rotational axis, with north positive
X-Y Plane: Earth's equator
X-Axis: directed toward the prime (Greenwich) meridian
Y- Axis: 90 deg from X and Z, completing a right-handed system

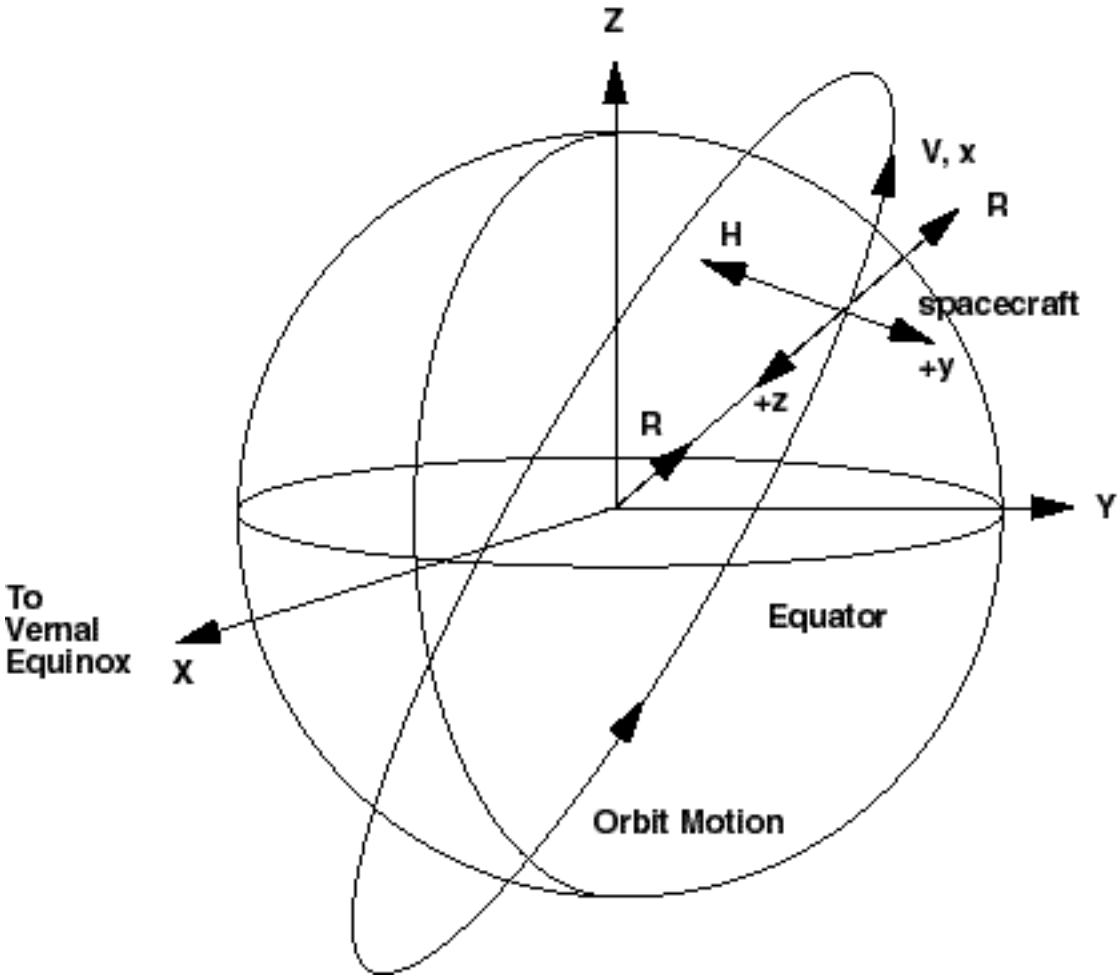
***Figure 6-1. Earth-Centered Rotating (ERC) Coordinates***




**EARTH-CENTERED INERTIAL (ECI) COORDINATES**

Origin: Center of the Earth
Z-Axis: along Earth's rotational axis, with north positive
X-Y Plane: Earth's equator
X-Axis: directed toward the vernal equinox
Y- Axis: 90 deg from X and Z, completing a right-handed system

***Figure 6-2. Earth Centered Inertial (ECI) Coordinates***




X, Y, Z are inertial coordinates. x, y, z are orbital coordinates, defined as follows:

Origin: Spacecraft Center of Mass
x-z plane: Spacecraft orbital plane

+y (pitch) - Axis: oriented normal to the orbit plane with positive sense opposite to that of the orbit's angular momentum vector H.

+z (yaw) - Axis: positively oriented earthward parallel to the satellite radius vector R from the spacecraft center of mass to the center of the Earth

+x (roll) - Axis: positively oriented in the direction of orbital flight completing an orthogonal triad with y and z.

***Figure 6-3. Relationship Between Earth-Centered Inertial (ECI) Coordinates and Orbital Coordinates***

# Transform from ECI to ECR Coordinates

**NAME:** **PGS_CSC_ECItoECR( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>
#include <PGS_TD.h>

PGSt_SMF_status
PGS_CSC_ECItoECR(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posvelECI[][6],
        PGSt_double         posvelECR[][6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitoecr (numvalues,asciiutc,offsets,posveleci,posvelecr)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posveleci(6,*)
        double precision    posvelecr(6,*)
```

**DESCRIPTION**: This function rotates an array of 6-vectors from ECI (J2000) coordinates to ECR (of date) coordinates. The rotation is done in 4 parts: precession, nutation, Earth rotation about the nutated axis, and polar motion (correction from the rotational North to geographic North).

**INPUTS**:

*Table 6-178. PGS_CSC_ECItoECR Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1972-01-01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posvelECI[6] | vector (position and velocity) in J2000 to be transformed to ECR of date | | | |
| posvelECI[0].. | x position | meters | | |
| posvelECI[1] | y position | meters | | |
| posvelECI[2].. | z position | meters | | |
| posVelECI[3] | x velocity | meters/ second | | |
| posVelECI[4] | y velocity | meters/ second | | |
| posvelECI[5] | z velocity | meters/ second | | |

**OUTPUTS**:

*Table 6-179. PGS_CSC_ECItoECR Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posvelECR[0] | vector after being transformed to ECR of date - x position | meters | | |
| posvelECR[1] | vector after being transformed to ECR of date - y position | meters | | |
| posvelECR[2] | vector after being transformed to ECR of date - z position | meters | | |
| posvelECR[3] | vector after being transformed to ECR of date - x velocity | meters/second | | |
| posvelECR[4] | vector after being transformed to ECR of date - y velocity | meters/second | | |
| posvelECR[5] | vector after being transformed to ECR of date - z velocity | meters/second | | |

**RETURNS**:

*Table 6-180. PGS_CSC_ECItoECR Returns (1 of 2)*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_BAD_TRANSFORM_VALUE | Invalid ECItoECR transformation |
| PGSCSC_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available input time |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |

*Table 6-180. PGS_CSC_ECItoECR Returns (2 of 2)*

| Return | Description |
|---|---|
| PGSCSC_W_PREDICTED_UT1 | Status of UT1–UTC correction is predicted |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posvelECI[ARRAY_SIZE][6] =
                            {
                              {0.5,0.75,0.90,0.3,0.2,0.8},
                              {0.65,1.2,3.65,0.1,3.2,1,7},
                              {0.98,2.6,4,78,0.2,1.5,0.9}
                            };
PGSt_double       posvelECR[ARRAY_SIZE][6];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECItoECR(numValues,asciiUTC,offsets,
                                posvelECI,posvelECR)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitoecr
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posvelECI(6,3)
double precision  posvelECR(6,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      asciiutc = '2002-07-27T11:04:57.987654Z'
      numvalues = 3

      DO 10 cnt1 = 1,6
         DO 10 cnt2 = 1,3
               posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 CONTINUE

      returnstatus = pgs_csc_ecitoecr(numValues, asciiutc,
     &                        offsets, posvelECI, posvelECR )

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

**NOTES**: Users not needing to transform velocity can supply floating point numbers equal to zero for the last three components of each input vector. The Tool cannot transform velocity, however, without correct values for the position. Note that to avoid generating absuredly large velocities for distant objects, no velocity transformation is performed for points more than 500,000,000 m from Earth center.

UTC is: Coordinated Universal Time

J2000 is Julian Date 2451545.0

See Section 6.3.4.8 Coordinate System Conversion Tool Notes.

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

**REFERENCES:**

The Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

**REQUIREMENTS**: PGSTK–1050

# Transform from ECR to ECI Coordinates

**NAME:** **PGS_CSC_ECRtoECI( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECRtoECI(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posvelECR[][6],
        PGSt_double         posvelECI[][6])
```

FORTRAN:

```
include'PGS_CSC_4.f'
include'PGS_TD_3.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function
pgs_csc_ecrtoeci(numvalues,asciiutc,offsets,posvelecr,posveleci)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posvelecr(6,*)
        double precision    posveleci(6,*)
```

**DESCRIPTION**: This function rotates an array of 6-vectors from ECR (of date) coordinates to ECI (J2000) coordinates. The rotation is done in 4 parts: polar motion (correction from the geographic North to rotational North), rotation about the true rotation, nutation to the mean of date axis axis, and precession to J2000).

**INPUTS**:

*Table 6-181. PGS_CSC_ECRtoECI Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1972-01-01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posvelECR[6] | vector (position and velocity) in ECR | | | |
| posvelECR[0].. | position | meters | | |
| posvelECR[2] | | | | |
| posvelECR[3].. | velocity | meters/ seconds | | |
| posvelECR[5] | | | | |

**OUTPUTS**:

*Table 6-182. PGS_CSC_ECRtoECI Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posvelECI[6] | vector after being transformed to J2000 | | | |
| posvelECI[0].. | position | meters | | |
| posvelECI[2] | | meters | | |
| posvelECI[3].. | velocity | meters/second | | |
| posvelECI[5] | | meters/second | | |

**RETURNS**:

*Table 6-183. PGS_CSC_ECRtoECI Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_BAD_TRANSFORM_VALUE | Invalid ECItoECR transformation |
| PGSCSC_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available input time |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSCSC_W_PREDICTED_UT1 | Status of UT1–UTC correction is predicted |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posvelECR[ARRAY_SIZE][6] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       posvelECI[ARRAY_SIZE][6];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECRtoECI(numValues,asciiUTC,offsets,
                                posvelECR,posvelECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer      pgs_csc_ecrtoeci
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posveleci(6,3)
double precision  posvelecr(6,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

asciiutc = '2002-07-27T11:04:57.987654Z'
numvalues = 3
```

```
      do 10 cnt1 = 1,6
          do 10 cnt2 = 1,3
              posvelecr(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_ecrtoeci (numValues, asciiutc,
                                       offsets, posvelecr,
                                       posveleci)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

**NOTES:** Users not needing to transform velocity can supply floating point numbers equal to zero for the last three components of each input vector. The Tool cannot transform velocity, however, without correct values for the position. Note that to avoid generating absuredly large velocities for disant objects, no velocity transformation is performed for points more than 500,000,000 m from Earth center.

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes.

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

**REFERENCES:**

The Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

**REQUIREMENTS**: PGSTK–1050

# Convert from ECR to Geodetic Coordinates

**NAME:** **PGS_CSC_ECRtoGEO( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECRtoGEO(
        PGSt_double     posECR[3],
        char            *earthEllipsTag,
        PGSt_double     *longitude,
        PGSt_double     *latititude,
        PGSt_double     *altitude);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_CSC_4.f'

integer function
pgs_csc_ecrtogeo(posecr,earthellipstag,longitude,latitude,height)
        double precision    posecr(3)
        character*49        earthellipstag
        double precision    longitude
        double precision    latitude
        double precision    altitude
```

**DESCRIPTION**: This function converts from ECR to geodetic coordinates.

**INPUTS**:

***Table 6-184. PGS_CSC_ECRtoGEO Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posECR[3] | geocentric position | meters | N/A | N/A |
| EarthEllipsTag | Earth model used | N/A | N/A | N/A |

**OUTPUTS**:

***Table 6-185. PGS_CSC_ECRtoGEO Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| latitude | geodetic latitude | radians | -pi/2 | pi/2 |
| longitude | longitude | radians | -pi | pi |
| altitude | altitude | meters | -.1* Earth | sky's the limit |

**RETURNS**:

*Table 6-186. PGS_CSC_ECRtoGEO Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_TOO_MANY_ITERS | Normal Iteration Count exceeded—could indicate inconsistent units for Spacecraft and Earth data, or corrupted Earth Axis values |
| PGSCSC_W_INVALID_ALTITUDE | Spacecraft underground—probably indicates bad input data |
| PGSCSC_W_SPHERE_BODY | Using a spherical Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater then 0.01 |
| PGSCSC_W_DEFAULT_EARTH_MODEL | Uses default Earth model |
| PGSCSC_E_BAD_EARTH_MODEL | The equatorial or polar radius is negative or zero OR the radii define a prolate Earth |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES**:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       longitude
PGSt_double       latitude
PGSt_double       altitude
char              earthEllipsTag[50],
PGSt_double       posECR[3] = {1000.5,64343.56,34343.92}
char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

strcpy(earthEllipsTag,"WGS84");

returnStatus = PGS_CSC_ECRtoGEO(posECR[3],earthEllipsTag,
                                longitude,latitude,
                                altitude);
if(returnStatus != PGS_S_SUCCESS)
  {
      PGS_SMF_GetMsg(&returnStatus,err,msg);
      printf("\nERROR: %s",msg);
  }
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecrtogeo
integer           returnstatus
```

```
double precision  longitude
double precision  latitude
double precision  altitude
character*49      earthellipstag,
double precision  posecr(3)
character*33      err
character*241     msg

data posECR/1000.5,64343.56,34343.92/
earthellipstag = 'WGS84'

returnstatus = pgs_csc_ecrtogeo(posecr,earthellipstag,
                                longitude,latitude,altitude)

if(returnstatus .ne. pgs_s_success) then
    returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
    write(*,*) err, msg
endif
```

**NOTES**: The Earth axes will be accessed from the earthfigure.dat.file. The input must always be in meters and should never be a unit vector.

**REQUIREMENTS**: PGSTK–0930, PGSTK–1050

# Convert from Geodetic to ECR Coordinates

**NAME:** **PGS_CSC_GEOtoECR( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GEOtoECR(
        PGSt_double     longitude,
        PGSt_double     latitude,
        PGSt_double     altitude,
        char            *earthEllipsTag,
        PGSt_double     posECR[3]);
```

FORTRAN:

```
include'PGS_SMF.f'
include'PGS_CSC_4.f'

integer function
pgs_csc_geotoecr(longitude,latitude,altitude,earthellipstag,posecr)
        double precision    longitude
        double precision    latitude
        double precision    altitude
        character*49        earthellipstag
        double precision    posecr(3)
```

**DESCRIPTION**: This tool converts a geodetic latitude and longitude to ECR (Earth Centered Rotating) coordinates.

**INPUTS**:

***Table 6-187. PGS_CSC_GEOtoECR Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| longitude | longitude | radians | -pi | pi |
| latitude | latitude | radians | -pi/2 | pi/2 |
| altitude | altitude | meters | -.1* radius | N/A |
| earthellipstag | Earth model used | N/A | N/A | N/A |

**OUTPUTS**:

*Table 6-188. PGS_CSC_GEOtoECR Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posECR | ECR rectangular coordinates | meters | -100,000,000 (usually each component will be in range [-10,000,000, +10,000,000 m ] but function will work for Geosynchronous cases, e.g. ) | 100,000,000 |

**RETURNS**:

*Table 6-189. PGS_CSC_GEOtoECR Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success case |
| PGSCSC_W_DEFAULT_EARTH_MODEL | The default Earth model is used because a correct one was not specified |
| PGSCSC_W_SPHERICAL_BODY | Using a spherical Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater then 0.01 |
| PGSCSC_W_INVALID_ALTITUDE | An invalid altitude was specified |
| PGSCSC_E_BAD_EARTH_MODEL | The equatorial or polar radius is negative or zero OR the radii define a prolate Earth |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES**:

C:

```
PGSt_SMF_status   returnStatus
PGSt_double       longitude
PGSt_double       latitude
PGSt_double       altitude
char              earthEllipsTag[50],
PGSt_double       posECR[3]
char              err[PGS_SMF_MAX_MNEMONIC_SIZE];
char              msg[PGS_SMF_MAX_MSG_SIZE];

longitude = 0.45;
latitude = 1.34;
altitude = 5000.0;
strcpy(earthEllipsTag,"WGS84");
```

```
returnStatus = PGS_CSC_GEOtoECR(longitude,latitude,altitude,
                                earthEllipsTag,posECR);
if(returnStatus != PGS_S_SUCCESS)
  {
      PGS_SMF_GetMsg(&returnStatus,err,msg);
      printf("\nERROR: %s",msg);
  }
```

FORTRAN:

```
implicit none

integer           pgs_csc_geotoecr
integer           returnstatus
double precision  longitude
double precision  latitude
double precision  altitude
character*49      earthellipstag,
double precision  posecr(3)
character*33      err
character*241     msg

longitude = 0.45
latitude = 1.34
altitude = 5000
earthellipstag = 'WGS84'

returnstatus = pgs_csc_geotoecr(longitude,latitude,altitude,
                                earthellipstag,posecr)

if(returnstatus .ne. pgs_s_success) then
    returnstatus = pgs_smf_getmsg(returnstatus,err,msg)
    write(*,*) err, msg
endif
```

**NOTES**: NONE

**REQUIREMENTS**: PGSTK–0930, PGSTK–1050

# Transform from ECI Frame to Spacecraft Reference Frame

**NAME:** **PGS_CSC_ECItoSC( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECItoSC(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posECI[][3],
        PGSt_double         posSC[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitosc(spacecraftTag,numvalues,asciiutc,offsets,poseci,possc)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    poseci(3,*)
        double precision    possc(3,*)
```

**DESCRIPTION**: Transforms vector in ECI coordinate system to vector in Spacecraft coordinate system. If a unit vector is input, only its direction is transformed. If a vector in meters is input, it is first corrected for the displacement between Earth center and spacecraft location and then rotated into spacecraft coordinates.

**INPUTS**:

*Table 6-190. PGS_CSC_ECItoSC Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1961–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posECI | coordinates<br>or unit vector components in ECI reference frame | meter | N/A | N/A |

**OUTPUTS**:

*Table 6-191. PGS_CSC_ECItoSC Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posSC | coordinates or unit vector components in spacecraft reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-192. PGS_CSC_ECItoSC Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |
| PGSEPH_E_NO_DATA_REQUESTED | Both orb and att flags are set to false |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posECI[ARRAY_SIZE][3] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       posSC[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ECItoSC(PGSd_TRMM,numValues,asciiUTC,
                               offsets,posECI,posSC)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitosc
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  poseci(3,3)
double precision  possc(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cnt1 = 1,3
    do 10 cnt2 = 1,3
        posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_ecitosc(PGSd_TRMM, numValues,
   asciiutc,
                                  offsets, poseci, possc)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

**NOTES**: Points are checked to make sure they are not subterranean, but no other visibility check is performed (such as line–of–sight).

Next the function checks the input vector to see if it is a unit vector. If so, it is assumed that the user wishes only to transform its direction. If not, it is assumed that the vector locates some point of interest (for example, a TDRSS satellite, or a lookpoint). Thus, for that case a translation to the spacecraft center is performed first and then a rotation. Aberration correction is also performed in both cases, except in the second case, for points within 120 m of spacecraft center. Vectors to such points are not aberrated. This cutoff is imposed on the supposition that anyone wishing to transform a point within 120 m of the spacecraft center could be dealing with an alignment, glint, or other spacecraft-related problem, in which case there is no aberration. For the purposes of this function, a vector is a unit vector if its magnitude is between 0.99999 and 1.00001.

**TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

# Transform Between Spacecraft and ECI Reference Frames

**NAME:** **PGS_CSC_SCtoECI( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SCtoECI(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posSC[][3],
        PGSt_double         posECI[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_sctoeci(spacecraftTag,numvalues,asciiutc,offsets,possc,poseci)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    possc(3,*)
        double precision    poseci(3,*)
```

**DESCRIPTION**: Transforms vector in Spacecraft coordinate system to vector in ECI coordinate system. If a unit vector is input, it is simply rotated to ECI coordinates. If a vector in meters in input, it is rotated to ECI axes and then translated from having its origin at the spacecraft center to having its origin at Earth center.

**INPUTS**:

*Table 6-193. PGS_CSC_SCtoECI Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1961–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posSC | coordinates or unit vector components in SC reference frame | meters | N/A | N/A |

**OUTPUTS**:

*Table 6-194. PGS_CSC_SCtoECI Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posECI | coordinates or unit vector components in ECI reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-195. PGS_CSC_SCtoECI Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input |
| PGSEPH_E_NO_DATA_REQUESTED | Both orb and att flags are set to false |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posSC[ARRAY_SIZE][3] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       posECI[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_SCtoECI(PGSd_TRMM,numValues,
                               asciiUTC,offsets, posSC,
                               posECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer                 pgs_csc_sctoeci
integer                 returnstatus
integer                 numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  possc(3,3)
double precision  poseci(3,3)
integer                 cnt1
integer                 cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              posveleci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_sctoeci (PGSd_TRMM,numValues,
                                      asciiutc,offsets,possc,
                                      poseci)

      if (returnstatus .ne. pgs_s_success) then
           pgs_smf_getmsg(returnstatus, err, msg)
           write(*,*) err, msg
      endif
```

**NOTES**: This function first checks the input vector to see if it is a unit vector. If so, it is assumed that the user wishes only to transform its direction. If not, it is assumed that the vector locates some point of interest (for example, a TDRSS satellite, or a lookpoint). For that case a rotation to ECI axes is performed first, and then a translation to the Earth center. An aberration correction is also made if the input is a unit vector or is in meters and represents a point more than 120 m from spacecraft center. This cutoff is imposed on the supposition that anyone wishing to transform a point within 120 m of the spacecraft center could be dealing with an alignment, glint, or other spacecraft-related problem, in which case there is no aberration. For the purposes of this function, a vector is a unit vector if its magnitude is between 0.99999 and 1.00001

Certain checks are performed in the case of translation to ensure that the transformed point is not below the Earth's surface; other visibility checks (such as line–of–sight) are not performed.

**TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

# Transform from Spacecraft Frame to Orbital Frame

**NAME:** **PGS_CSC_SCtoORB( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SCtoORB(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posSC][3],
        PGSt_double         posORB[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_sctoorb(spacecraftTag,numvalues,asciiutc,offsets,possc,posorb)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    possc(3,*)
        double precision    posorb(3,*)
```

**DESCRIPTION**: Transforms vector in Spacecraft reference frame to a vector in Orbital reference frame.

**INPUTS**:

*Table 6-196. PGS_CSC_SCtoORB Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1961–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posSC | coordinates or unit vector components in SC reference frame | meters | N/A | N/A |

**OUTPUTS**:

*Table 6-197. PGS_CSC_SCtoORB Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posORB | coordinates or unit vector components in Orbital reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-198. PGS_CSC_SCtoORB Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posSC[ARRAY_SIZE][3] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       posORB[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_SCtoORB(pgsd_trmm,numvalues,asciiutc,
                               offsets,possc,posORB)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_sctoorb
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  possc(3,3)
double precision  posorb(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cbt1 = 1,3
    do 10 cnt2 = 1,3
        possc(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_sctoorb(pgsd_trmm,numvalues,
   asciiutc,
                                  offsets, possc, posorb)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

**NOTES**: **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

# Transform from Orbital Frame to Spacecraft Frame

**NAME:** **PGS_CSC_ORBtoSC( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ORBtoSC(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         posORB][3],
        PGSt_double         posSC[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_orbtosc(spacecrafttag,numvalues,asciiutc,offsets,posorb,possc)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    posorb(3,*)
        double precision    possc(3,*)
```

**DESCRIPTION**: Transforms vector from Orbital reference frame to a vector in Spacecraft reference frame.

**INPUTS**:

*Table 6-199. PGS_CSC_ORBtoSC Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1960–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| posORB | coordinates or unit vector components in Orbital reference frame | meters | N/A | N/A |

**OUTPUTS**:

*Table 6-200. PGS_CSC_ORBtoSC Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posSC | coordinates or unit vector components in spacecraft reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-201. PGS_CSC_ORBtoSC Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_double       posORB[ARRAY_SIZE][3] =
                              {
                                {0.5,0.75,0.90,0.3,0.2,0.8},
                                {0.65,1.2,3.65,0.1,3.2,1,7},
                                {0.98,2.6,4,78,0.2,1.5,0.9}
                              };
PGSt_double       posSC[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus = PGS_CSC_ORBtoSC(PGSd_TRMM,numValues,asciiUTC,
                               offsets, posORB, posSC)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_orbtosc
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  posorb(3,3)
double precision  possc(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

do 10 cnt1 = 1,3
    do 10 cnt2 = 1,3
        posorb(cnt1,cnt2) = 100 * cnt1 * cnt2
```

```
10 continue

   asciiutc = '1991-07-27T11:04:57.987654Z'
   numvalues = 3

   returnstatus = pgs_csc_orbtosc(pgsd_trmm,numvalues,
   asciiutc,
                                  offsets, posorb, possc)

   if (returnstatus .ne. pgs_s_success) then
       pgs_smf_getmsg(returnstatus, err, msg)
       write(*,*) err, msg
   endif
```

**NOTES:** **TIME ACRONYMS:**

UTC is: Coordinted Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

# Transform from ECI Frame to Orbital Frame

**NAME:** **PGS_CSC_ECItoORB( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ECItoORB(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         positionECI[][3],
        PGSt_double         positionORB[][3])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EHP_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_ecitoorb(spacecraftTag,numvalues,asciiutc,offsets,positioneci,
                                        positionorb)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    positioneci(3,*)
        double precision    positionorb(3,*)
```

**DESCRIPTION**: Transforms vector in ECI coordinate system to vector in Orbital coordinate system. If a unit vector is input only its direction is changed. If a vector in meters is input, it is first translated from the Earth centered system to a spacecraft centered origin, and then rotated to orbital coordinate axes.

**INPUTS**:

*Table 6-202. PGS_CSC_ECItoORB Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1961–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| positionECI | coordinates or unit vector components in ECI reference frame | meters | N/A | N/A |

**OUTPUTS**:

*Table 6-203. PGS_CSC_ECItoORB Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| positionORB | coordinates or unit vector components in orbital reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-204. PGS_CSC_ECItoORB Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                            {3600.0,7200.0,10800.0};
PGSt_double       positionORB[ARRAY_SIZE][3] =
                            {
                              {0.5,0.75,0.90,0.3,0.2,0.8},
                              {0.65,1.2,3.65,0.1,3.2,1,7},
                              {0.98,2.6,4,78,0.2,1.5,0.9}
                            };
PGSt_double       positionORB[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus =
PGS_CSC_ECItoORB(PGSd_TRMM,numValues,asciiUTC,
                                offsets, positionECI,
                                positionORB)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_ecitoorb
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  positioneci(3,3)
double precision  positionorb(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              positioneci(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_ecitoorb(pgsd_trmm,numvalues,
                                      asciiutc,offsets,
                                      positioneci, positionorb)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

**NOTES:** **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

# Transform from Orbital Frame to ECI Frame

**NAME:** **PGS_CSC_ORBtoECI( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ORBtoECI(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         positionORB[][3],
        PGSt_double         positionECI[][3])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_SMF.f'

integer function pgs_csc_orbtoeci(spacecraftTag,numvalues,asciiutc,
                                  offsets,positionorb,positioneci)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    positionorb(3,*)
        double precision    positioneci(3,*)
```

**DESCRIPTION**: Transforms vector in Orbital coordinate system to vector in ECI coordinate system. If a unit vector is input it is simply rotated from Orbital to ECI axes. If a vector in meters is input, it is first rotated from Orbital to ECI axes and then translated from the system referenced at spacecraft center to the system referenced at Earth center.

**INPUTS**:

*Table 6-205. PGS_CSC_ORBtoECI Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1961–01–01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| positionORB | coordinates or unit vector components in Orbital reference frame | meters | N/A | N/A |

**OUTPUTS**:

*Table 6-206. PGS_CSC_ORBtoECI Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| positionECI | coordinates or unit vector components in ECI reference frame | meters | N/A | N/A |

**RETURNS**:

*Table 6-207. PGS_CSC_ORBtoECI Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSSC_W_BELOW_SURFACE | Vector magnitude indicates subsurface location specified |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_TIME_FMT_ERROR | Format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Value error in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                            {3600.0,7200.0,10800.0};
PGSt_double       positionORB[ARRAY_SIZE][3] =
                            {
                              {0.5,0.75,0.90,0.3,0.2,0.8},
                              {0.65,1.2,3.65,0.1,3.2,1,7},
                              {0.98,2.6,4,78,0.2,1.5,0.9}
                            };
PGSt_double       positionECI[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30.123211Z");
returnStatus =
PGS_CSC_ORBtoECI(PGSd_TRMM,numValues,asciiUTC,offsets,
                 positionORB,positionECI)

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_orbtoeci
integer           returnstatus
integer           numvalues
character*27      asciiutc
double precision  offsets(3)
double precision  positionorb(3,3)
double precision  positioneci(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/
```

```
      do 10 cnt1 = 1,3
          do 10 cnt2 = 1,3
              positionorb(cnt1,cnt2) = 100 * cnt1 * cnt2

   10 continue

      asciiutc = '1991-07-27T11:04:57.987654Z'
      numvalues = 3

      returnstatus = pgs_csc_orbtoeci(pgsd_trmm,numvalues,
                                      asciiutc,offsets,
                                      positionorb,positioneci)

      if (returnstatus .ne. pgs_s_success) then
          pgs_smf_getmsg(returnstatus, err, msg)
          write(*,*) err, msg
      endif
```

**NOTES:** **TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.1 (TAI-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS**: PGSTK–1050

### 6.3.4.10 Coordinate System Conversion—Other Tools

These tools provide other location and orientation information to the user.

## Get Sub–Satellite Point Position and Velocity

---

**NAME:** **PGS_CSC_SubSatPoint( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SubSatPoint(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        char                earthEllipsTag[50],
        PGSt_boolean        velFlag,
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_double         altitude[],
        PGSt_double         velSub[][3])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_MEM_7.f'

integer function
pgs_csc_subsatpoint(spacecrafttag,numvalues,asciiutc,offsets,

        earthellipstag,velflag,latitude,longitude,
                                    altitude,velsub)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        character*49        earthellipstag
        integer             velflag
        double precision    latitude(*)
```

| | |
|---|---|
| double precision | longitude(*) |
| double precision | altitude(*) |
| integer | velsub(3,*) |

**DESCRIPTION:** This tool finds the latitude, longitude, and altitude of the subsatellite points at the input times/offsets and, optionally, returns North and East components of each subsatellite point. The third component returned for each subsatellite point, when velocity is requested, is the rate of change of the spacecraft altitude off the Earth ellipsoid (as would be measured by a Doppler radar altimiter, ignoring terrain).

**INPUTS:**

***Table 6-208. PGS_CSC_SubSatPoint Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input offset times | N/A | 0 | any |
| asciiUTC | timesstart UTC time in CCSDS ASCII Time Code (A or B format) | N/A | 1979–06–30 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC + offset is between Min and Max values | |
| earthEllipsTag | tag selecting Earth ellipsoid model (default is WGS84) | N/A | N/A | N/A |
| velFlag | flag indicating whether to return the velocity of the subsatellite points | N/A | PGS_FALSE | PGS_TRUE |

**OUTPUTS:**

***Table 6-209. PGS_CSC_SubSatPoint Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| latitude | array of subsatellite point geodetic latitudes | radians | -pi/2 | pi/2 |
| longitude | array of subsatellite point longitudes | radians | -pi | pi |
| altitude | array of spacecraft altitudes | m | 250000 | 10000000 |
| velSub[0] | North component of the subsatellite point velocity on the ellipsoid | m/s | -7000 | 7000 |
| velSub[1] | East component of the subsatellite point velocity on the ellipsoid | m/s | -7000 | 7000 |
| velSub[2] | rate of change of spacecraft altitude relative to nadir on the ellipsoid | m/s | -200 | 200 |

**RETURNS:**

***Table 6-210. PGS_CSC_SubSatPoint Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_ERROR_IN_SUBSATPT | An error occurred in computing at least one subsatellite point |
| PGSCSC_W_PREDICTED_UT1 | At least one of the values obtained from the utcpole.dat file is 'predicted' |
| PGSCSC_W_PROLATE_BODY | Using a prolate Earth model |
| PGSCSC_W_SPHERE_BODY | Using a spherical Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater than 0.01 |
| PGSCSC_W_DEFAULT_EARTH_MODEL | Default Earth model was used |
| PGSCSC_W_ZERO_JACOBIAN_DET | Jacobian determinant is close to zero |
| PGSCSC_E_BAD_ARRAY_SIZE | numValues (and array size) is less than zero |
| PGSMEM_E_NO_MEMORY | No memory available to allocate vectors |
| PGSTD_E_SC_TAG_UNKNOWN | Invalid spacecraft tag |
| PGSEPH_E_BAD_EPHEM_FILE_HEADER | No spacecraft ephemeris files had reasonable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No spacecraft ephemeris files could be found for input |
| PGSTD_E_TIME_FMT_ERROR | Format error in input asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | Error in one of time values in asciiUTC |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for at least one of the input times/offsets—a linear approximation was used to obtain the leapsec value |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGSTD_E_BAD_EARTH_MODEL | The equatorial or polar radius is negative or zero OR the radii define a prolate Earth |
| PGS_E_TOOLKIT | Something unexpected happened—execution aborted |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_tag          spacecraftTag = PGSd_EOS_AM;
PGSt_integer      numValues;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                  {3600.0,7200.0,10800.0};
char              earthEllipsTag[50];
PGSt_boolean      velFlag = PGS_TRUE;
PGSt_double       latitude[ARRAY_SIZE];
```

```
PGSt_double       longitude[ARRAY_SIZE];
PGSt_double       altitude[ARRAY_SIZE];
PGSt_double       velSub[ARRAY_SIZE][3];
PGSt_integer      counter;

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
strcpy(earthEllipsTag,"WGS84");

returnStatus = PGS_CSC_SubSatPoint(spacecraftTag,numValues,
                                   asciiUTC,offsets,
                                   earthEllipseTag,velFlag,
                                   latitude,longitude,
                                   altitude,velSub);

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter <= numValues)
{
    printf("Offset:  %lf   Latitude:  %lf   Longitude:  %lf
            Altitude:  %lf",   offset[counter],
            latitude[counter],  longitude[counter],
            altitude[counter]);
    printf("Velocity of subsatellite point
            (North,East,altitude): %lf, %lf, %lf"  " m/s",
             velSub[counter][0], velSub[counter][1],
             velSub[counter][2]);

             counter++;
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_subsatpoint
integer           array_size
integer           spacecrafttag
integer           numvalues
character*27      asciiutc
double precision  offsets(array_size)
character*49      earthellipstag
integer           velflag
```

```
      double precision  latitude(array_size)
      double precision  longitude(array_size)
      double precision  altitude(array_size)
      double precision  velsub(3,array_size)
      integer           returnstatus
      integer           counter

      data offsets/3600.0,7200.0,10800.0/
      data earthellipstag/'WGS84'/,velflag/PGS_TRUE/
      array_size = 3
      numvalues = array_size
      spacecrafttag = pgsd_eos_am
      asciiutc = '1991-01-01T11:29:30'

      returnstatus = pgs_csc_subsatpoint(spacecrafttag,numvalues,
                                         asciiutc,offsets,
                                         earthellipsetag,velflag,
                                         latitude,longitude,
                                         altitude,velsub)

      if(returnstatus .ne. pgs_s_success) go to 90
      write(6,*) asciiutc
      do 40 counter = 0,numvalues,1
          write(6,*)offsets(counter), latitude(counter),
                    longitude(counter), altitude(counter),
                    velsub(1,counter), velsub(2,counter),
                    velsub(3,counter)

   40 continue

   90 write(6,99)returnstatus

   99 format('ERROR:',I50)
```

**NOTES:**

If an error occurs during computation for one or more input times but does not necessarily affect all input times, latitude, longitude, altitude, and velocity values of PGSd_GEO_ERROR_VALUE are returned for the input times where the error occurred. An indication that an error occurred in this tool is returned in the returnStatus value, and a description of the error is returned in the corresponding message.

If an invalid earthEllipsTag is input, the program will use the WGS84 Earth model by default.

The option to obtain velocity is controlled by setting the velocity flag velFlag to either PGS_TRUE or PGS_FALSE. If velFlag is PGS_FALSE, all components of velSub will be set to zero. If the velocity is not needed it is recommended to use PGS_FALSE to speed the execution of the code.

The horizontal velocity calculated in function PGS_CSC_SubSatPointVel() is that of a mathematical point on the Earth at (nominal) spacecraft nadir, and not that of any material object. It is orthogonal to nadir, so is suitable as a descriptor of ground track but not for Doppler work.

The third (vertical) component of velocity is useful for Doppler work at nadir, but Doppler velocity along ANY look vector (not just nadir) is provided in the lookpoint algorithm in the function PGS_CSC_GetFOV_Pixel( ).

The condition PGSCSC_W_ZERO_JACOBIAN_DET is not expected to occur. Its appearance would indicate that the geometry is singular: the altitude of the spacecraft is zero or the spacecraft is exactly at the north or south pole, for example.

**TIME ACRONYMS:**

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REFERENCES FOR TIME:**

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

**REQUIREMENTS:** PGSTK–0930, PGSTK–1060

# Get Times of Earth Point in Fixed Field of View

**NAME:** **PGS_CSC_Earthpt_FixedFOV( )**

**SYNOPSIS:**

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CSC_Earthpt_FixedFOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        char                *earthEllipsTag,
        PGSt_double         latitude,
        PGSt_double         longitude,
        PGSt_double         altitude,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[3],
        PGSt_double         perimFOV_vectors[][3],
        PGSt_boolean        inFOVflag[],
        PGSt_double         sctoEarthptVec[][3])
```

FORTRAN:

```
include'PGS_TD_3.f'
include'PGS_CSC_4.f'
include'PGS_EPH_5.f'
include'PGS_MEM_7.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function pgs_csc_earthpt_fixedfov(numvalues,asciiutc,offsets,
                                spacecrafttag,earthellipstag,latitude,
                                longitude,altitude,numfovperimvec,
                                infovvector,perimfov_vectors,
                                infovflag,sctoearthptvec)

        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             spacecrafttag
        character*49        earthellipstag
```

| | |
|---|---|
| double precision | latitude |
| double precision | longitude |
| double precision | altitude |
| integer | numfovperimvec |
| double precision | infovvector(3) |
| double precision | perimfov_vectors(3,*) |
| integer | infovflag(*) |
| double precision | sctoearthptvec(3,*) |

**DESCRIPTION:** For each time value, the tool, using the FOV description, returns a flag or flags indicating if the Earth point of given latitude, longitude and altitude is in the FOV, and the vector to that point from the SC in SC coordinates.

**INPUTS:**

***Table 6-211. PGS_CSC_Earthpt_FixedFOV Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of time gridpoints | N/A | 0 | any |
| asciiUTC | UTC start time | N/A | 1972-01-01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| earthEllipsTag | Earth model used | N/A | N/A | N/A |
| latitude | latitude of Earth point | radians | -pi/2 | +pi/2 |
| longitude | longitude of Earth point | radians | -2*pi | +2*pi |
| altitude | altitude of Earth point | meters | -50000 | 100000 |
| numFOVperimVec | number of vectors defining FOV perimeter | N/A | 3 | any |
| inFOVvector | vector in FOV—preferably near the center in SC coordinates | N/A | N/A | N/A |
| perimFOV_vectors | vectors in SC coords defining FOV's; MUST be sequential around FOV; the middle dimension must be exactly the same as numFOVperimVec because of the way the array dimensioning works in the function | N/A | N/A | N/A |

**OUTPUTS:**

*Table 6-212. PGS_CSC_Earthpt_FixedFOV Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| inFOVflag | PGS_TRUE if Earth point is in FOV—see notes | n/a | n/a | n/a |
| sctoEarthptVec | vector to Earth point in SC coords—returned normalized | meters | -1 | 1 |

**RETURNS:**

*Table 6-213. PGS_CSC_Earthpt_FixedFOV Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | Location is below surface |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_BAD_INITIAL_TIME | Initial time is incorrect |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSCSC_W_DEFAULT_EARTH_MODEL | The default Earth model is used because a correct one was not specified |
| PGSCSC_W_DATA_FILE_MISSING | The data file earthfigure.dat is missing |
| PGSCSC_W_SPHERICAL_BODY | Using a spherical Earth model |
| PGSCSC_W_PROLATE_BODY | Using a prolate Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater then 0.01 |
| PGSCSC_E_INVALID_ALTITUDE | An invalid altitude was specified |
| PGSCSC_E_NEG_OR_ZERO_RAD | The equatorial or polar radius is negative or zero |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSCSC_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSCSC_W_PREDICTED_UT1 | Status of UT1–UTC correction is predicted |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |
| PGSCSC_E_INVALID_FOV_DATA | FOV perimeter vectors are invalid |
| PGSCSC_E_FOV_TOO_LARGE | FOV specification outside algorithmic limits |
| PGSCSC_E_INVALID_EARTH_PT | One of the Earth point vectors was zero |
| PGSCSC_W_ZERO_PIXEL_VECTOR | Instrument pixel vector of zero length |
| PGSCSC_W_BAD_EPH_FOR_PIXEL | Ephemeris Data missing for some pixels |

**EXAMPLES:**

C:

```
#define   ARRAY_SIZE   3
#define   PERIMVEC_SIZE 4
```

```
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_integer      numValues;
PGSt_double       latitude;
PGSt_double       longitude;
PGSt_double       altitude;
PGSt_integer      numFOVperimVec;
PGSt_double       inFOVvector[3] =
                        { {0.0,0.0,100.0},
                        };
PGSt_double       perimFOV_vectors[PERIMVEC_SIZE][3]=
                        { {100.0,100.0,100.0},
                          {-100.0,100.0,100.0},
                          {-100.0,-100.0,100.0},
                          {100.0,-100.0,100.0} };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       sctoEarthptVec[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");
altitude = 10000.0;
latitude = 0.32;
longitude = 2.333;
returnStatus =
PGS_CSC_Earthpt_FixedFOV(numValues,asciiUTC,offsets,
                    PGSd_TRMM,"WGS84",latitude,
                    longitude,altitude,numFOVperimVec,
                    inFOVvector,perimFOV_vectors,inFOVflag,
                    sctoEarthptVec)
if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_earthpt_fixedfov
integer           returnstatus
integer           numvalues
character*27      startutc
double precision  offsets(3)
double precision  latitude
double precision  longitude
double precision  altitude
integer           numfovperimvec
double precision  infovvector(3)
double precision  perimfov_vectors(3,4)
integer           infovflag(3)
double precision  sctoearthptvec(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

perimfov_vectors(1,1) = 100.0
perimfov_vectors(2,1) = 100.0
perimfov_vectors(3,1) = 100.0

perimfov_vectors(1,2) = -100.0
perimfov_vectors(2,2) = 100.0
perimfov_vectors(3,2) = 100.0

perimfov_vectors(1,3) = -100.0
perimfov_vectors(2,3) = -100.0
perimfov_vectors(3,3) = 100.0

perimfov_vectors(1,4) = 100.0
perimfov_vectors(2,4) = -100.0
perimfov_vectors(3,4) = 100.0

infovvector(1) = 0.0
infovvector(2) = 0.0
infovvector(3) = 100.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4
altitude = 10000.0
latitude = 0.32
longitude = 2.333
```

```
returnstatus =
pgs_csc_earthpt_fixedfov(numvalues,startutc,offsets,
                     PGSd_TRMM,'WGS84',latitude,
                     longitude,altitude,numfovperimvec,
                     infovvector,perimfov_vectors,
                     infovflag,sctoearthptvec)

if (returnstatus .ne. pgs_s_success) then
    pgs_smf_getmsg(returnstatus, err, msg)
    write(*,*) err, msg
endif
```

**NOTES:** At each time, the tool determines if the Earth point at (latitude, longitude, altitude) is in the FOV, setting inFOVflag = PGS_TRUE if so, else PGS_FALSE. The vector from SC to Earth point is also returned, whether or not the Earth point is in the FOV, and even if it is on the far side of the Earth. Test for the spacecraft to Earth point being equal to 1.0e50 to avoid processing Earth points that could not be determined because of one or more errors in the transformation.

The FOV is always specified and fixed in SC coordinates. numFOVperimVec should be at least 3. The tool determines if the Earth point lies within the perimeter defined by the vectors perim–FOVvectors[][3]. The first index in C (last in FORTRAN) runs around the perimeter and must be sequential. If the altitude is unknown use zero.

The vector inFOVvector[3] must be defined in SC coordinates and must lie within the FOV. It is necessary for the user to supply a vector within the FOV because on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside the FOV, which outside. If the vector is well centered in the FOV, the algorithm will be faster.

The vectors "perimFOV_vectors[][3]" defining the FOV perimeter can be in clock or counter–clockwise sequence. If the FOV perimeter vectors are supplied out of order, the algorithm will run but the results are unpredictable. The input vectors need not be normalized but must not be zero.

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS:** PGSTK–1090

# Get Times of Earth Point in Field of View

**NAME:** **PGS_CSC_Earthpt_FOV( )**

**SYNOPSIS:**

C:

```
#include <PGS_TD.h>
#include <PGS_CSC.h>
#include <PGS_EPH.h>
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_CSC_Earthpt_FOV(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_tag            spacecraftTag,
        char                *earthEllipsTag,
        PGSt_double         latitude,
        PGSt_double         longitude,
        PGSt_double         altitude,
        PGSt_integer        numFOVperimVec,
        PGSt_double         inFOVvector[][3],
        void                *perimFOV_vectors,
        PGSt_boolean        inFOVflag[],
        PGSt_double         sctoEarthptVec[][3])
```

FORTRAN:

```
include'PGS_TD_3.f'
include'PGS_CSC_4.f'
include'PGS_EPH_5.f'
include'PGS_MEM_7.f'
include'PGS_TD.f'
include'PGS_SMF.f'

integer function pgs_csc_earthpt_fov(numvalues,asciiutc,offsets,
                                     spacecrafttag,earthellipstag,latitude,
                                     longitude,altitude,numfovperimvec,
                                     infovvector,perimfov_vectors,
                                     infovflag,sctoearthptvec)

        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        integer             spacecrafttag
        character*49        earthellipstag
```

| | |
|---|---|
| double precision | latitude |
| double precision | longitude |
| double precision | altitude |
| integer | numfovperimvec |
| double precision | infovvector(3,*) |
| double precision | perimfov_vectors(3,*,*) |
| integer | infovflag(*) |
| double precision | sctoearthptvec(3,*) |

**DESCRIPTION:** For each time value, the tool, using the FOV description, returns a flag or flags indicating if the Earth point of given latitude, longitude and altitude is in the FOV, and a unit vector to that point from the SC in SC coordinates.

**INPUTS:**

***Table 6-214. PGS_CSC_Earthpt_FOV Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of time gridpoints | N/A | 0 | any |
| asciiUTC | UTC start time | N/A | 1972-01-01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+offset is between asciiUTC Min and Max values | |
| spacecraftTag | unique spacecraft identifier | N/A | N/A | N/A |
| earthEllipsTag | Earth model used | N/A | N/A | N/A |
| latitude | latitude of Earth point | radians | -pi/2 | +pi/2 |
| longitude | longitude of Earth point | radians | -2*pi | +2*pi |
| altitude | altitude of Earth point | meters | -50000 | 100000 |
| numFOVperimVec | number of vectors defining FOV perimeter | N/A | 3 | any |
| inFOVvector | vector in FOV—preferably near the center in SC coordinates | N/A | N/A | N/A |
| perimFOV_vectors | vectors in SC coords defining FOV's; MUST be sequential around FOV; the middle dimension must be exactly the same as numFOVperimVec because of the way the array dimensioning works in the function | N/A | N/A | N/A |

**OUTPUTS:**

*Table 6-215. PGS_CSC_Earthpt_FOV Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| inFOVflag | PGS_TRUE if Earth point is in FOV—see notes | n/a | n/a | n/a |
| sctoEarthptVec | vector to Earth point in SC coords—returned normalized | meters | -1 | 1 |

**RETURNS:**

*Table 6-216. PGS_CSC_Earthpt_FOV Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_BELOW_SURFACE | Location is below surface |
| PGSCSC_W_BAD_TRANSFORM_VALUE | One or more values in transformation could not be determined |
| PGSTD_E_BAD_INITIAL_TIME | Initial time is incorrect |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction available for input time |
| PGSCSC_W_DEFAULT_EARTH_MODEL | The default Earth model is used because a correct one was not specified |
| PGSCSC_W_DATA_FILE_MISSING | The data file earthfigure.dat is missing |
| PGSCSC_W_SPHERICAL_BODY | Using a spherical Earth model |
| PGSCSC_W_PROLATE_BODY | Using a prolate Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater then 0.01 |
| PGSCSC_E_INVALID_ALTITUDE | An invalid altitude was specified |
| PGSCSC_E_NEG_OR_ZERO_RAD | The equatorial or polar radius is negative or zero |
| PGSMEM_E_NO_MEMORY | No memory is available to allocate vectors |
| PGSCSC_E_BAD_ARRAY_SIZE | Incorrect array size |
| PGSCSC_W_PREDICTED_UT1 | Status of UT1–UTC correction is predicted |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input times |
| PGSCSC_E_INVALID_FOV_DATA | FOV perimeter vectors are invalid |
| PGSCSC_E_FOV_TOO_LARGE | FOV specification outside algorithmic limits |
| PGSCSC_E_INVALID_EARTH_PT | One of the Earth point vectors was zero |
| PGSCSC_W_ZERO_PIXEL_VECTOR | Instrument pixel vector of zero length |
| PGSCSC_W_BAD_EPH_FOR_PIXEL | Ephemeris Data missing for some pixels |

**EXAMPLES:**

C:

```
#define   ARRAY_SIZE   3
#define   PERIMVEC_SIZE 4
PGSt_SMF_status   returnStatus;
char              asciiUTC[28];
```

```
PGSt_double       offsets[ARRAY_SIZE] =
                        {3600.0,7200.0,10800.0};
PGSt_integer      numValues;
PGSt_double       latitude;
PGSt_double       longitude;
PGSt_double       altitude;
PGSt_integer      numFOVperimVec;
PGSt_double       inFOVvector[ARRAY_SIZE][3] =
                        { {0.0,0.0,100.0},
                          {0.0,0.0,200.0},
                          {0.0,0.0,300.0}
                        };
PGSt_double
      perimFOV_vectors[ARRAY_SIZE][PERIMVEC_SIZE][3]=
                        { {100.0,100.0,100.0},
                          {-100.0,100.0,100.0},
                          {-100.0,-100.0,100.0},
                          {100.0,-100.0,100.0},
                          {200.0,200.0,200.0},
                          {-200.0,200.0,200.0},
                          {-200.0,-200.0,200.0},
                          {200.0,-200.0,200.0},
                          {300.0,200.0,200.0},
                          {-200.0,300.0,200.0},
                          {-200.0,-300.0,300.0},
                          {300.0,-200.0,200.0},
                        };
PGSt_boolean      inFOVflag[ARRAY_SIZE];
PGSt_double       sctoEarthptVec[ARRAY_SIZE][3];

numValues = ARRAY_SIZE;
numFOVperimVec = PERIMVEC_SIZE;
strcpy(asciiUTC,"1995-06-21T11:29:30.123211Z");
altitude = 10000.0;
latitude = 0.32;
longitude = 2.333;
returnStatus =
PGS_CSC_Earthpt_FOV(numValues,asciiUTC,offsets,
                    PGSd_TRMM,"WGS84",latitude,
                    longitude,altitude,numFOVperimVec,
                    inFOVvector,perimFOV_vectors,inFOVflag,
                    sctoEarthptVec)
if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
```

```
        take appropriate
         action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_earthpt_fov
integer           returnstatus
integer           numvalues
character*27      startutc
double precision  offsets(3)
double precision  latitude
double precision  longitude
double precision  altitude
integer           numfovperimvec
double precision  infovvector(3,4)
double precision  perimfov_vectors(3,4,3)
integer           infovflag(3)
double precision  sctoearthptvec(3,3)
integer           cnt1
integer           cnt2
character*33      err
character*241     msg

data offsets/3600.0, 7200.0, 10800.0/

perimfov_vectors(1,1,1) = 100.0
perimfov_vectors(2,1,1) = 100.0
perimfov_vectors(3,1,1) = 100.0

perimfov_vectors(1,2,1) = -100.0
perimfov_vectors(2,2,1) = 100.0
perimfov_vectors(3,2,1) = 100.0

perimfov_vectors(1,3,1) = -100.0
perimfov_vectors(2,3,1) = -100.0
perimfov_vectors(3,3,1) = 100.0

perimfov_vectors(1,4,1) = 100.0
perimfov_vectors(2,4,1) = -100.0
perimfov_vectors(3,4,1) = 100.0

perimfov_vectors(1,1,2) = 200.0
perimfov_vectors(2,1,2) = 200.0
perimfov_vectors(3,1,2) = 200.0

perimfov_vectors(1,2,2) = -200.0
perimfov_vectors(2,2,2) = 200.0
perimfov_vectors(3,2,2) = 200.0
```

```
perimfov_vectors(1,3,2) = -200.0
perimfov_vectors(2,3,2) = -200.0
perimfov_vectors(3,3,2) = 200.0

perimfov_vectors(1,4,2) = 200.0
perimfov_vectors(2,4,2) = -200.0
perimfov_vectors(3,4,2) = 200.0

perimfov_vectors(1,1,3) = 300.0
perimfov_vectors(2,1,3) = 300.0
perimfov_vectors(3,1,3) = 300.0

perimfov_vectors(1,2,3) = -300.0
perimfov_vectors(2,2,3) = 300.0
perimfov_vectors(3,2,3) = 300.0

perimfov_vectors(1,3,3) = -300.0
perimfov_vectors(2,3,3) = -300.0
perimfov_vectors(3,3,3) = 300.0

perimfov_vectors(1,4,3) = 300.0
perimfov_vectors(2,4,3) = -300.0
perimfov_vectors(3,4,3) = 300.0

infovvector(1,1) = 0.0
infovvector(1,2) = 0.0
infovvector(1,3) = 100.0

infovvector(2,1) = 0.0
infovvector(2,2) = 0.0
infovvector(2,3) = 200.0

infovvector(3,1) = 0.0
infovvector(3,2) = 0.0
infovvector(3,3) = 300.0

asciiutc = '1995-06-21T11:04:57.987654Z'
numvalues = 3
numfovperimvec = 4
altitude = 10000.0
latitude = 0.32
longitude = 2.333

returnstatus =
pgs_csc_earthpt_fov(numvalues,startutc,offsets,
                    PGSd_TRMM,'WGS84',latitude,
                    longitude,altitude,numfovperimvec,
                    infovvector,perimfov_vectors,
                    infovflag,sctoearthptvec)
```

```
if (returnstatus .ne. pgs_s_success) then
    pgs_smf_getmsg(returnstatus, err, msg)
    write(*,*) err, msg
endif
```

**NOTES:** At each time, the tool determines if the Earth point at (latitude, longitude, altitude) is in the FOV, setting inFOVflag = PGS_TRUE if so, else PGS_FALSE. The vector from SC to Earth point is also returned, whether or not the Earth point is in the FOV, and even if it is on the far side of the Earth. Test for the spacecraft to Earth point being equal to 1.0e50 to avoid processing Earth points that could not be determined because of one or more errors in the transformation.

The FOV is always specified in SC coordinates. For an instrument fixed to the SC, use the same FOV description always. For scanning instruments, user should provide the description appropriate to the scan instrument. numFOVperimVec should be at least 3. The tool determines if the Earth point lies within the perimeter defined by the vectors perim–FOVvectors[][][3]. The first index in C (last in FORTRAN) is the time offset index and the second must be sequential around the FOV perimeter. If the altitude is unknown use zero.

The vector inFOVvector[][3] must be defined in SC coordinates and must lie within the FOV. The last index in C, (first in FORTRAN) on these vectors is for X,Y, and Z, components in SC coordinates. It is necessary for the user to supply a vector within the FOV because on the surface of a sphere, a closed curve or "perimeter" does not have an inside nor outside, except by arbitrary definition; i.e., this vector tells the algorithm which part of sky is inside the FOV, which outside. If the vector is well centered in the FOV, the algorithm will be faster.

The vectors "perimFOV_vectors[][][3]" defining the FOV perimeter can be in clock or counter–clockwise sequence. If the FOV perimeter vectors are supplied out of order, the algorithm will run but the results are unpredictable. The input vectors need not be normalized but must not be zero.

See Section 6.3.4.8 Conversion System Coordinate Tool Notes

See Section 6.2.7.5.1 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REQUIREMENTS:** PGSTK–1090

# Estimate Refraction of Ray

**NAME:** **PGS_CSC_SpaceRefract( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_SpaceRefract(
        PGSt_double   spaceZenith,
        PGSt_double   altitude,
        PGSt_double   latitude,
        PGSt_double   *surfaceZenith,
        PGSt_double   *displacement)
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_spacerefract(spacezenith,altitude,
                                      latitude,surfacezenith, displacement)
        double precision      spacezenith
        double precision      altitude
        double precision      latitude
        double precision      surfacezenith
        double precision      displacement
```

**DESCRIPTION:** This function estimates the refraction of a ray incident from space or a line of sight from space to the Earth's surface based on the unrefracted zenith angle (most common algorithms, intended for ground based observation, require knowledge of the refracted, not the unrefracted zenith angle). The algorithm is suitable for:

a. approximate determination of the apparent Solar zenith angle from the true (geometrical, unrefracted) Solar zenith angle (obviously, also applicable to Lunar zenith angle, etc.)

b. correction of the viewing angle from space, to approximately remove the effects of refraction

The method is briefly indicated in the NOTES, q.v. for various caveats.

**INPUTS:**

*Table 6-217. PGS_CSC_SpaceRefract Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spaceZenith | unrefracted zenith angle | radians | 0 | pi/2 (90 deg) |
| altitude | altitude off the geoid | meters | -1000 | 50000 |
| latitude | latitude | radians | -pi/2 | pi/2 |

**OUTPUTS:**

*Table 6-218. PGS_CSC_SpaceRefract Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| surfaceZenith | refracted zenith angle | radians | 0 | n/a |
| displacement | displacement of the footpoint of ray | radians | 0 | ~0.01 |

**RETURNS:**

*Table 6-219. PGS_CSC_SpaceRefract Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGS_CSC_BAD_LAT | a latitude out of the range (- pi/2, pi/2) was entered |
| PGS_CSC_E_INVALID_ZENITH | a negative zenith angle was entered |
| PGSCSC_W_INVALID_ALTITUDE | Attempt to calculate refraction at point too far below Earth's surface |
| PGSCSC_W_BELOW_HORIZON | Attempt to calculate refraction of ray below horizon |

**EXAMPLES**:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double spaceZenith=0.4;
PGSt_double altitude=5000.0;
PGSt_double latitude= - 0.2 ;  **** not implemented at
                                    present ***
PGSt_double surfaceZenith;
PGSt_double displacement;

returnStatus = PGS_CSC_SpaceRefract(spaceZenith,altitude,
                                    latitude,&surfaceZenith,
                                    &displacement)

{
  ** test errors,
      take appropriate
```

```
        action **
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_spacerefract
integer           returnstatus
double precision  spacezenith
double precision  altitude
double precision  latitude
double precision  surfacezenith
double precision  displacement

data spacezenith /0.4/
data altitude /5000.0/
data latitude /-0.2/

returnstatus = pgs_csc_spacerefract(spacezenith,altitude,
                                    latitude,surfacezenith,
                                    displacement)

if(returnstatus .ne. pgs_s_success) go to 90
write(6,*) surfacezenith,displacement

90  write(6,99)returnstatus
99  format('ERROR:',I15)
```

**NOTES:**

This algorithm is intended as a mean-atmosphere approximation, valid for white light (for example, sunlight). Refraction is quite wavelength dependent, and in the atmosphere it will also depend strongly on local conditions (e.g., the weather). The present algorithm is intended to be a reasonable approximation such that to do better one would need local and, for large zenith angles, regional weather.

Caveat: The altitude is used ONLY to obtain the air pressure, which is then used to obtain the surface index of refraction. Users who employ an inflated Earth radius in geolocation should be especially careful to replace any derived altitude with the height in meters above the geoid before calling this function.

The method is based on the author's calculations, using a conservation law originally due to W. Chauvenet, for the important difference z0 - z', and an empirical refraction algorithm in Equation 3.283-1, p. 144, Astron. Almanac Supplement (U.S. Naval Observatory) to derive the less important displacement.

The (horizontal) displacement of the ray is in a vertical plane containing the ray and is in the sense that the actual (refracted) ray will meet the Earth d = (displacement)*Re meters from the geometrical (unrefracted) position, on the side towards the horizon.

```
 Outer Space  Here
      .
```

```
          . unrefracted ray
             .
refracted ray  .*
                  . * unrefracted ray
_____________________.__*_________________ Earth surface
                       d
```

the angle "displacement" is the angle that the displacement in meters "d" subtends at Earth center.

The following table exemplifies results at sea level, using a conversion of 6371000 m per radian on the displacement.

***Table 6-220. Altitude – Sea Level***

| Zenith Angle in Space (deg) | Zenith Angle at Surface (deg) | Refraction (deg) | Linear Displacement (meters) |
|---|---|---|---|
| 10.000000 | 9.997066 | 0.002934 | 0.549064 |
| 20.000000 | 19.993944 | 0.006056 | 1.222937 |
| 30.000000 | 29.990394 | 0.009606 | 2.221314 |
| 40.000000 | 39.986039 | 0.013961 | 3.982978 |
| 45.000000 | 44.983363 | 0.016637 | 5.464087 |
| 50.000000 | 49.980174 | 0.019826 | 7.725334 |
| 55.000000 | 54.976243 | 0.023757 | 11.398788 |
| 60.000000 | 59.971192 | 0.028808 | 17.845724 |
| 61.000000 | 60.969996 | 0.030004 | 19.696711 |
| 62.000000 | 61.968722 | 0.031278 | 21.816620 |
| 63.000000 | 62.967361 | 0.032639 | 24.256691 |
| 64.000000 | 63.965905 | 0.034095 | 27.080360 |
| 65.000000 | 64.964340 | 0.035660 | 30.366779 |
| 70.000000 | 69.954333 | 0.045667 | 58.380584 |
| 75.000000 | 74.938025 | 0.061975 | 136.072953 |
| 76.000000 | 75.933417 | 0.066583 | 166.728721 |
| 77.000000 | 76.928121 | 0.071879 | 207.384912 |
| 78.000000 | 77.921967 | 0.078033 | 262.469333 |
| 79.000000 | 78.914723 | 0.085277 | 338.977167 |
| 80.000000 | 79.906069 | 0.093931 | 448.379942 |
| 81.000000 | 80.895543 | 0.104457 | 610.332976 |
| 82.000000 | 81.882461 | 0.117539 | 860.316290 |
| 83.000000 | 82.865762 | 0.134238 | 1266.536004 |
| 84.000000 | 83.843713 | 0.156287 | 1970.638000 |
| 85.000000 | 84.813286 | 0.186714 | 2974.066487 |
| 86.000000 | 85.768718 | 0.231282 | 4858.394025 |
| 87.000000 | 86.697712 | 0.302288 | 8677.416632 |
| 88.000000 | 87.569758 | 0.430242 | 17538.457911 |
| 89.000000 | 88.295108 | 0.704892 | 41818.325388 |
| 90.000000 | 88.619113 | 1.380887 | 113429.256196 |

Note that the linear displacement at 88 degrees zenith angle is about 17.5 km—very substantial. Because of the very approximate atmosphere model, this number could vary by perhaps 25% depending on weather in

temperate and tropical regions; in the Arctic it would be considerably smaller. The displacement at 90 degrees incidence, over 113 km, is only suggestive and could easily vary by 50%.

The increments in latitude and longitude due to refraction are:

| **Direction** | **Value** |
|---|---|
| latitude ($\phi$) | dAng * cos($\psi$) |
| longitude ($\lambda$) | dAng * sin($\psi$)/cos($\phi$) |

where $\psi$ is the azimuth from PGS_CSC_ZenithAzimuth(). The expression for longitude is singular at the North and South poles and the user should avoid using it there, or within too close range. When |latitude| > $\pi$ - dAng, the point is so near the pole that the displacement of the ray can be assumed to be South at the North pole and North at the South pole; but when starting at either pole, the longitude (not its increment) must be found from -atan2(yray,xray) where (xray,yray,zray) are the components of the look vector in ECR. After calling PGS_CSC_SpaceRefract(), then, the user who is interested in the displacement in latitude and longitude needs to implement the equations above and, for the exceptional case at a pole, the alternate just explained: latitude = dAng, longitude = -atan2(yray,xray). The Toolkit software does not perform these operations, which are a user responsibility if the positional correction is desired.

The composition of the atmosphere was obtained from Allen's "Astrophysical Quantities, 2nd ed." (London, the Athlone Pre, 1976) p. 121, because the U.S. Standard Atmosphere (NOAA, 1976) is bone dry, which is unrealistic.

The atmosphere model is used only to get the index of refraction at sea level. The latitude dependence is that the sea level temperature and mean scale height are functions of latitude.

The calculations are based on the geometry of a spherical Earth. User may employ her/his favorite Earth radius to transform radians of displacement to meters. See also “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger, where the equation to transform displacement magnitude to North and East components is given.

**REQUIREMENTS:** PGSTK–0860, PGSTK-1080

# Get Field-of-View Footprint and Pixel Centers

**NAME:** **PGS_CSC_GetFOV_Pixel( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GetFOV_Pixel(
        PGSt_tag            spacecraftTag,
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        char                earthEllipsTag[50],
        PGSt_boolean        accurFlag,
        PGSt_double         pixelUnitvSC[][3],
        PGSt_double         offsetXYZ[][3],
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_double         pixelUnitvECR[][3],
        PGSt_double         slantRange[],
        PGSt_double         velocDoppl[])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_EPH_5.f'
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_getfov_pixel(spacecrafttag,numvalues,asciiutc,
                                      offsets, earthellipstag,accurflag,
                                      pixelunitvsc,offsetxyz,latitude,
                                      longitude,pixelunitvecr,
                                      slantrange,velocdoppl)
        integer             spacecrafttag
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        character*49        earthellipstag
        integer             accurflag
        double precision    pixelunitvsc(3,*)
        double precision    offsetxyz(3,*)
        double precision    latitude(*)
```

| | |
|---|---|
| double precision | longitude(*) |
| double precision | pixelunitvecr(3,*) |
| double precision | slantrange(*) |
| double precision | velocdoppl(*) |

**DESCRIPTION:** This function obtains the latitude and longitude of the intersection of a line of sight with the spheroidal Earth, the slant range from Spacecraft to look point, and the Doppler velocity along the line of sight. The ECR pixel vector is also returned; it can be used, for example, to determine the zenith angle of the line of sight. The line of sight is defined by a unit vector in the Spacecraft frame of reference and a time. (The unit vector along the line of sight is called a "look vector" in the sequel.)

The Doppler velocity is true, in the sense that it is relative to the Earth's surface.

**INPUTS:**

***Table 6-221. PGS_CSC_GetFOV_Pixel Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| spacecraftTag | spacecraft identifier | N/A | N/A | N/A |
| numValues | number of input time offsets (to use ASCII time with no offsets, set numValues =0 or set it =1 and make first [and only] offset = 0.0) | N/A | 0 | N/A |
| asciiUTC | UTC start time in CCSDS ASCII Time A or B format | N/A | 1972-01-01 | see NOTES |
| offsets | array of time offsets | SI seconds | Max and Min such that floating equivalent of asciiUTC+offset is between asciiUTC Min and Max values | |
| EarthEllipsTag | tag selecting Earth Ellipsoid model | N/A | N/A | N/A |
| accurFlag | flag to regulate accuracy | N/A | PGS_FALSE | PGS_TRUE |
| pixelUnitvSC | array of pixel unit vectors in SC coords | N/A | -1 | 1 |
| offsetXYZ | array of displacements of instrument boresight from SC nominal center in SC coordinates(see overall limit for length of this vector in "RETURNS" section) (offsetXYZ is used only when accurFlag == PGS_TRUE) | m | -120 | +120 |

**OUTPUTS:**

***Table 6-222. PGS_CSC_GetFOV_Pixel Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| latitude | latitude of the lookpoint | radians | -pi/2 | pi/2 |
| longitude | longitude of the lookpoint | radians | -pi | pi |
| pixelUnitvECR | ECR unit pixel vector | N/A | -1 | +1 |
| slantRange | slant range: SC to lookpoint | m | 0 | 100000 |
| velocDoppl | Doppler velocity of the look point (+ meaning "away") | m/s | -8000 | 8000 |

## RETURNS:

*Table 6-223. PGS_CSC_GetFOV_Pixel Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_W_MISS_EARTH | Look Vector fails to intersect Earth |
| PGSTD_E_SC_TAG_UNKNOWN | Invalid Spacecraft tag |
| PGSCSC_W_ZERO_PIXEL_VECTOR | Instrument pixel vector of zero length |
| PGSCSC_W_BAD_EPH_FOR_PIXEL | Ephemeris Data missing for some pixels |
| PGSCSC_W_INSTRUMENT_OFF_BOARD | Instrument offset from SC center is > 120 m which is considered unreasonably large (applicable only when accurFlag = PGS_TRUE) |
| PGSCSC_W_BAD_ACCURACY_FLAG | Accuracy Flag neither PGS_TRUE nor PGS_FALSE |
| PGSCSC_E_BAD_ARRAY_SIZE | The user has supplied a negative number of time offsets |
| PGSCSC_W_DEFAULT_EARTH_MODEL | Invalid EarthEllipsTag; WGS84 model used |
| PGSCSC_W_DATA_FILE_MISSING | A file such as the ephemeris, utcpole, Earth Model or leap seconds file is missing |
| PGSCSC_E_NEG_OR_ZERO_RAD | One of the Earth axes is zero or negative |
| PGSMEM_E_NO_MEMORY | Malloc operation for scratch memory failed |
| PGSTD_E_NO_LEAP_SECS | no leap seconds data available for input time |
| PGSTD_E_TIME_FMT_ERROR | format error in asciiUTC |
| PGSTD_E_TIME_VALUE_ERROR | value error in asciiUTC |
| PGSCSC_W_PREDICTED_UT1 | predicted UT1 value used |
| PGSCSC_E_NO_UT1_VALUE | no UT1 value available |
| PGS_E_TOOLKIT | Error in Toolkit—for example, inconsistent error message from a subordinate function |
| PGSEPH_E_BAD_EPHEM_FILE_HDR | No s/c ephem files had readable headers |
| PGSEPH_E_NO_SC_EPHEM_FILE | No s/c ephem files could be found for input |

## EXAMPLES:

C:

```
#include <PGS_CSC.h>
char        asciiUTC[28] = "1994-01-15T12:21:33.9939Z";
PGSt_tag          spacecraftTag = PGSd_EOS_AM;
char              EarthEllipsTag[50] = "WGS84";
PGSt_double       offsets[4]  = {0.0,0.1,2.0,30.0};
PGSt_double       pixelUnitvSC[4][3];
PGSt_double       offsetXYZ[4][3];
PGSt_integer      numValues = 4;
PGSt_boolean      accurFlag = PGS_FALSE;

PGSt_double       latitude[4];
PGSt_double       longitude[4];
PGSt_double       velocDoppl[4];
PGSt_double       slantRange[4];
PGSt_double       pixelUnitvECR[4][3];
```

```
PGSt_SMF_status   returnStat;
PGSt_SMF_status   code;
char              msg[240];
char              mnemonic[31];
int               i;
int               jj;

for (i=0;i<4;i++)
     for(jj=0;jj<3 ;++jj)
         offsetXYZ[i][jj] = 0.0;

/** initialize pixel unit vectors
   All but the 3rd case hit Earth; to miss Earth reverse
   the last component of any other one **/

pixelUnitvSC[0][0] = 0.03;
pixelUnitvSC[0][1] = 0.12;
pixelUnitvSC[0][2] = 0.08;

pixelUnitvSC[1][0] =  -0.2;
pixelUnitvSC[1][1] =  0.12;
pixelUnitvSC[1][2] =  0.6;

/**This case will display error**/

pixelUnitvSC[2][0] =  -0.0;
pixelUnitvSC[2][1] =  0.00;
pixelUnitvSC[2][2] =  0.0;

pixelUnitvSC[3][0] =  -0.2;
pixelUnitvSC[3][1] =  -0.12;
PixelUnitvSC[3][2] =  0.6;

returnStat = PGS_CSC_GetFOV_Pixel(spacecraftTag,numValues,
                                  asciiUTC,offsets,
                                  EarthEllipsTag,accurFlag,
                                  pixelUnitvSC,offsetXYZ,
                                  latitude,longitude,
                                  pixelUnitvECR,santRange,
                                  velocDoppl);

printf("  Toolkit return value:  %d\n\n",returnStat);

PGS_SMF_GetMsg(&code,mnemonic,msg);
printf("  Return %s: %s\n\n",mnemonic,msg);

printf(" accurFlag ==  %d  Earth Tag == %s  ECR Pixels:\n"
        "%15.11lg    %15.11lg    %15.11lg\n"
        "%15.11lg    %15.11lg    %15.11lg\n "
        "%15.11lg    %15.11lg    %15.11lg\n"
```

```
        "%15.11lg    %15.11lg    %15.11lg\n",
        accurFlag,EarthEllipsTag,
        pixelUnitvECR[0][0],pixelUnitvECR[0][1],
            pixelUnitvECR[0][2],
        pixelUnitvECR[1][0],pixelUnitvECR[1][1],
            pixelUnitvECR[1][2],
        pixelUnitvECR[2][0],pixelUnitvECR[2][1],
            pixelUnitvECR[2][2],
        pixelUnitvECR[3][0],pixelUnitvECR[3][1],
            pixelUnitvECR[3][2]);

/**  Test for some variable like latitude =
    PGSd_GEO_ERROR_VALUE before further processing to avoid
    processing pixels that missed Earth or had zero pixel
    vector. In multi-pixel processing, results from good and
    bad pixels can be distinguished only by answers being
    PGSd_GEO_ERROR_VALUE; in single pixel processing return
    status indicates any error **/

if(returnStatus != PGS_S_SUCCESS)
{
  /**  print results - latitude, longitude, etc.; test
      errors, take appropriate action **/
}
```

FORTRAN:

```
implicit none

parameter(numPixels = 4)
integer           pgs_csc_getfov_pixel
integer           spacecrafttag
integer           numvalues
character*27      asciiutc
double precision  offsets(numPixels)
character*49      earthellipstag
integer           accurflag
double precision  pixelUnitvSC(3,numPixels)
double precision  offsetXYZ(3,numPixels)
double precision  latitude(numPixels)
double precision  longitude(numPixels)
double precision  pixelUnitvECR(3,numPixels)
double precision  slantRange(numPixels)
double precision  velocDoppl(numPixels)
character*33      err
character*241     msg
```

```
      data offsets/360.0, 720.0, 1080.0, 1600.0/
      asciiutc = '1991-07-27T11:04:57.987654Z'
      spacecrafttag = PGSd_EOS_AM

      do 1 jj = 1,3
      do 1 i = 1,4
         offsetXYZ(jj,i) = 0.0;

1      continue

!

!     This puts instrument at the nominal SC center

1     For example, to put instrument on a 20 m boom fore of

!     SC center, make  offsetXYZ(1,i) = 20.0 for each i

!    initialize pixel unit vectors

!    All but the 3rd case hit Earth; to miss Earth reverse the

!    last component of any other one

      pixelUnitvSC(1,1) = 0.03;
      pixelUnitvSC(2,1) = 0.12;
      pixelUnitvSC(3,1) = 0.08;

      pixelUnitvSC(1,2) =  -0.2;
      pixelUnitvSC(2,2) =  0.12;
      pixelUnitvSC(3,2) =  0.6;

!     This case will display error

      pixelUnitvSC(1,3) =  -0.0;
      pixelUnitvSC(2,3) =  0.00;
      pixelUnitvSC(3,3) =  0.0;

      pixelUnitvSC(1,4) =  -0.2;
      pixelUnitvSC(2,4) =  -0.12;
      pixelUnitvSC(3,4) =  0.6;

      returnstatus = pgs_csc_getfov_pixel(spacecrafttag,numvalues,
      >                                   asciiutc,offsets,
      >                                   earthellipstag,
      >                                   accurflag,pixelUnitvSC,
      >                                   offsetXYZ,latitude,
      >                                   longitude,pixelUnitvECR,
      >                                   slantRange,velocDoppl)

!    Print output values
```

```
!               Test for some variable like latitude = 1.0e50 before further

!               processing to avoid processing pixels that missed Earth or had

!               zero pixel vector

                   if (returnstatus .ne. pgs_s_success) then
                       pgs_smf_getmsg(returnstatus, err, msg)
                       write(*,*) err, msg
                   endif
```

**NOTES:** An accuracy flag is required, allowing two accuracy levels:

Normal or PGS_FALSE

- do ECI to ECR transformation at moment of taking data
- consider instrument axis to pass through nominal center of spacecraft

High or PGS_TRUE

- do ECI to ECR transformation with approximate allowance for Earth rotation during the light travel time (spherical Earth approximation.) This will slow the calculation slightly.
- user must supply vector offsetXYZ that represents the displacement in meters of the instrument boresight from nominal spacecraft center. (Only the part of the displacement orthogonal to the look vector will have an effect.) Users invoking the High Accuracy option but wishing not to take advantage of this feature should supply zeros for the components of offsetXYZ.

The maximum error in omitting this calculation is approximately as follows for a worst case of a spacecraft at 700 km altitude, crossing the equator and looking E or W:

***Table 6-224. Error due to Earth Motion in Time of Flight of Light***

| Nadir Angle (deg) | Slant Range (km) | Worst Case Error (m) if accurFlag = PGS_FALSE |
|---|---|---|
| 0 | 700 | 1.1 |
| 30 | 830 | 1.3 |
| 40 | 945 | 1.5 |
| 50 | 1200 | 1.9 |
| 55 | 1410 | 2.1 |
| 60 | 1770 | 2.7 |
| 64 | 2440 | 3.7 |

The nature of the error is a smooth distortion such that points near either the East or the West limb would be assigned a longitude slightly to the West in comparison with points near nadir. The effect could be somewhat exaggerated, for some orbits, in terms of illumination changes near the terminator.

Caution: The user is advised that the spacecraft ephemeris refers to the nominal center of the spacecraft. The displacements of individual instruments relative to the center of the spacecraft are taken into account herein through the vector offsetXYZ. When the flag "accurFlag" is set to PGS_TRUE, the user should specify the instrument coordinates relative to spacecraft center (in meters) with this vector. It WILL be used by the present function, so if the user does not actually wish to employ it, then offsetXYZ must be set to zero (all three components). If "accurFlag" is set to PGS_FALSE, the displacement is ignored.

**TIME ACRONYMS:**

UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

See Section 6.2.6.3 Spacecraft Tags Definition File

**REFERENCES FOR TIME:**

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac. See also “Theoretical Basis of the SDP Toolkit Geolocation Package for the ECS Project”, Document 445-TP-002-002, May 1995, by P. Noerdlinger.

**REQUIREMENTS:** PGSTK–0930, PGSTK–1080, PGSTK–1083,

## Precesses a Vector Between TDB Julian Date and J2000 Coordinates

**NAME:** **PGS_CSC_precs2000( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_precs2000(
        PGSt_integer        threeOr6,
        PGSt_double         jedTDB[2],
        PGSt_boolean        frwd,
        PGst_double         posVel[])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_TD.4.f'

integer function pgs_csc_precs2000(threeor6,jedtdb,fwrd,posvel)
        integer             threeor6
        double precision    jedtdb(2)
        integer             frwd
        double precision    posvel(6)
```

**DESCRIPTION:** This tool precesses a vector from Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Celestial Coordinates of date in Barycentric Dynamical Time (TDB).

**INPUTS:**

***Table 6-225. PGS_CSC_precs2000 Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| jedTDB[2] | TBD (Barycentric Dynamical Time) as a Julian Date to or from which the vector is to be processed | days | ANY | ANY |
| frwd | flag for sense of precession:<br>PGS_TRUE if precessing from J2000 to jedTDB<br>PGS_FALSE if precessing from jedTDB to J2000 | T/F | N/A | N/A |
| posVel | vector (position and velocity) in final reference frame:<br>posvel[0-2] position<br>posvel[3-5] velocity | <br><br>m<br>m/s | <br><br>ANY<br>ANY | <br><br>ANY<br>ANY |

## OUTPUTS:

*Table 6-226. PGS_CSC_precs2000 Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posVel | vector (position and velocity) in final reference frame: | | | |
| | posvel[0-2] position | m | ANY | ANY |
| | posvel[3-5] velocity | m/s | ANY | ANY |

## RETURNS:

*Table 6-227. PGS_CSC_precs2000 Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_BAD_ARRAY_SIZE | The size of the vector is not either 3 or 6 |
| PGSCSC_E_BAD_DIRECTION_FLAG | The value of the direction flag is not either PGS_TRUE or PGS_FALSE |

## EXAMPLES:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       posVel[6]={6400000.0,-5000000.0,40000.0,
                             4000.0,7000.0,-6000.0};

** precess the vector **

returnStatus = PGS_CSC_precs2000(6,jedTDB,PGS_TRUE,posVel);

** the input vector "posVel" has been overwritten with the
   precessed value **
```

FORTRAN:

```
implicit none

integer           pgs_csc_precs2000
integer           returnstatus
integer           threeor6
double precision  jedtdb(2)
double precision  posvel(6)

data jedtdb/2449720.5,0.25/
data posvel/6400000.0,-5000000.0,40000.0,4000.0,7000.0,-
            6000.0/

threeor6 = 6
```

```
returnstatus = pgs_csc_nutate2000(threeor6,jedtdb,frwd,
                                  posvel)
```

```
!      the input vector "posvel" has been overwritten with the precessed value
```

**NOTES:**

This function is a simplified version of PGS_CSC_precs3or6(). This function is specific to the case of precessing to or from the epoch of J2000. The various coefficients used are the constants that result for this epoch.

This function produces an output vector that overwrites the input vector. The code was kept this way to preserve its heritage. The user is cautioned that her/his input vector will be therefore be altered by this function. The underlying rotation functions do not have this property.

**TIME ACRONYMS:**

TDB is: Barycentric Dynamical Time

**JULIAN DATES:**

Format:

Toolkit Julian dates are kept as an array of two real (high precision) numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and return a Julian date will return the Julian date in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert the input date (internal) to the above format. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but will not check and the format of the output may not be what is expected if any other format is used for the input.

Meaning:

Toolkit "Julian dates" are all based on UTC. A Julian date in any other "time" (e.g., TAI, TDT, UT1, etc.) is based on the difference between that "time" and the equivalent UTC time (differences range in magnitude from 0 seconds to about a minute).

**REFERENCES FOR TIME:**

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

**REQUIREMENTS:** PGSTK–0930, PGSTK–1050

# Nutate State Vector Between True of Date and Mean of Date

**NAME:** **PGS_CSC_nutate2000( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_nutate2000(
        PGSt_integer        threeOr6,
        PGSt_double         jedTDB[2],
        PGSt_double         dvnut[4],
        PGSt_boolean        frwd,
        PGst_double         posVel[])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'

integer function pgs_csc_nutate2000(threeor6,jedtdb,dvnutfwrd,posvel)
        integer             threeor6
        double precision    jedtdb(2)
        double precision    dvnut(4)
        double precision    frwd
        double precision    posvel(*)
```

**DESCRIPTION:** This tool transforms a vector under nutation from Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Celestial Coordinates of date.

**INPUTS:**

***Table 6-228. PGS_CSC_nutate2000 Inputs (1 of 2)***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| threeOr6 | chooses a 3 or 6 dimensional vector to nutate | N/A | N/A | N/A |
| jedTDB | TBD (Barycentric Dynamical Time) as a Julian Date to or from which the vector is to be nutated (this variable is generally referred to in Toolkit code as jedTDB | days | ANY | ANY |
| dvnut | the two nutation angles and their rates, output from “PGS_CSC_wahr2” (this variable is generally referred to in Toolkit code as dvnut) | rad/s | -1.e-11 | 1.e-11 |
| posVel | vector (position and velocity) in initial reference frame:<br>posvel[0-2] position<br>posvel[3-5] velocity | <br>m<br>m/s | <br>ANY<br>ANY | <br>ANY<br>ANY |

*Table 6-228. PGS_CSC_nutate2000 Inputs (2 of 2)*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| **frwd** | flag for sense of nutation:<br>PGS_TRUE if nutating fromTrue of Date at<br>jedTDB to Mean of Date<br>PGS_FALSE if nutating from Mean of Date<br>to True of Date at jedTDB | T/F | N/A | N/A |

**OUTPUTS:**

*Table 6-229. PGS_CSC_nutate2000 Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posVel | vector (position and velocity) in final reference frame:<br>posvel[0-2] position<br>posvel[3-5] velocity | <br>m<br>m/s | <br>ANY<br>ANY | <br>ANY<br>ANY |

**RETURNS:**

*Table 6-230. PGS_CSC_nutate2000 Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSCSC_E_BAD_ARRAY_SIZE | The size of the vector is not either 3 or 6 |
| PGSCSC_E_BAD_DIRECTION_FLAG | The value of the direction flag is not either PGS_TRUE or PGS_FALSE |

**EXAMPLES:**

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       dvnut[4];
PGSt_double       posVel[6]={6400000.0,-5000000.0,40000.0,
                             4000.0,7000.0,-6000.0};

**  get the nutation angles and rates **

PGS_CSC_wahr2(jedTDB,dvnut);

** nutate the vector **

returnStatus = PGS_CSC_nutate2000(6,jedTDB,dvnut,PGS_TRUE,
                                  posVel);
```

```
** the input vector "posVel" has been overwritten with the
   nutated value **
```

FORTRAN:

```
implicit none

integer           pgs_csc_nutate2000
integer           returnstatus
integer           threeor6
double precision  jedtdb(2)
double precision  dvnut
double precision  posvel(6)

data jedtdb/2449720.5,0.25/
data posvel/6400000.0,-5000000.0,40000.0,4000.0,7000.0,
            -6000.0/

threeor6 = 6

!          get the nutation angles and rates

returnstatus = pgs_csc_wahr2(jedtdb,dvnut)

!          nutate the vector

returnstatus = pgs_csc_nutate2000(threeor6,jedtdb,dvnut,
                                  frwd,posvel)

!          the input vector "posvel" has been overwritten with the nutated
   value
```

**NOTES:**

Purpose: The case of transforming a vector from J2000 to True of Date, requires first procession and then nutation. The intermediate system, processed but not nutated, is the Mean of Date system. With the direction flag at PGS_TRUE, this function transforms a vector (position and velocity) from Mean of Date to True of Date. True of date has its Z axis along the Earth's true angular velocity and the X axis is toward the true equinox of date-the intersection of the equator perpendicular to Z with the ecliptic. Mean of date is arranged similarly, but ignoring nutation, so its pole has a constant angle to the ecliptic, along which its X axis moves at a constant rate.

In the opposite case, with the direction flag at PGS_FALSE, i.e. in going from arbitrary epoch to J2000, this function carries the vector from True of Date to Mean of Date, after which it must be precessed to J2000 by the function PGS_CSC_precs2000().

This code was modified so it now takes either a 3 or 6 dimensional vector. When 6 dimensions are used, they must be in the order (position, velocity) because the transformation of velocity is slightly different. This function produces an output vector that overwrites the input vector. The code was

kept this way to preserve its heritage. The user is cautioned that her/his input vector will therefore be altered by this function. The underlying rotation functions do not have this property.

**TIME ACRONYMS:**

TDB is: Barycentric Dynamical Time

**JULIAN DATES:**

Format:

Toolkit Julian dates are kept as an array of two real (high precision) numbers (C: PGSt_double, FORTRAN: DOUBLE PRECISION). The first element of the array should be the half integer Julian day (e.g., N.5 where N is a Julian day number). The second element of the array should be a real number greater than or equal to zero AND less than one (1.0) representing the time of the current day (as a fraction of that (86400 second) day. This format allows relatively simple translation to calendar days (since the Julian days begin at noon of the corresponding calendar day). Users of the Toolkit are encouraged to adhere to this format to maintain high accuracy (one number to track significant digits to the left of the decimal and one number to track significant digits to the right of the decimal). Toolkit functions that do NOT require a Julian type date as an input and return a Julian date will return the Julian date in the above mentioned format. Toolkit functions that require a Julian date as an input and do NOT return a Julian date will first convert the input date (internal) to the above format. Toolkit functions that have a Julian date as both an input and an output will assume the input is in the above described format but will not check and the format of the output may not be what is expected if any other format is used for the input.

Meaning:

Toolkit "Julian dates" are all based on UTC. A Julian date in any other "time" (e.g., TAI, TDT, UT1, etc.) is based on the difference between that "time" and the equivalent UTC time (differences range in magnitude from 0 seconds to about a minute).

**REFERENCES FOR TIME:**

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

**REQUIREMENTS:** PGSTK–0914, PGSTK–0930, PGSTK–1050

# Transform from ECI J2000 to ECI True of Date Coordinates

**NAME:** **PGS_CSC_J2000toTOD( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_J2000toTOD(
        PGSt_integer        threeOr6,
        PGSt_double         secTAI93
        PGSt_double         posvelECI[6],
        PGSt_double         posvelTOD[6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_SMF.f'

integer function pgs_csc_j2000totod(threeor6,sectai93,posveleci,
>                                   posveltod)
        integer             threeor6
        double precision    sectai93
        double precision    posveleci(*)
        double precision    posveltod(*)
```

**DESCRIPTION**: This function transforms from ECI (J2000) coordinates to TOD (true of date) coordinates.

**INPUTS:**

***Table 6-231. PGS_CSC_J2000toTOD.c Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| threeOr6 | dimension of input vector | N/A | 3 | 6 |
| secTAI93 | TOD time | seconds | | |
| posvelECI[] | Vector (position and possibly velocity) in ECI J2000 | | | |
| posvelTOD[0] | x position | meters | | |
| posvelTOD[1] | y position | meters | | |
| posvelTOD[2] | z position | meters | | |
| posvelTOD[3] | x velocity | m/s | | |
| posvelTOD[4] | y velocity | m/s | | |
| posvelTOD[5] | z velocity | m/s | | |

**OUTPUTS:**

*Table 6-232. PGS_CSC_J2000to.TOD.c Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posvelTOD[6] | Vector (position and possibly velecity) in ECI TOD | | | |
| posvelECI[0] | x position | meters | | |
| posvelECI[1] | y position | meters | | |
| posvelECI[2] | z position | meters | | |
| posvelECI[3] | x velocity | m/s | | |
| posvelECI[4] | y velocity | m/s | | |
| posvelECI[5] | z velocity | m/s | | |

**RETURNS:**

*Table 6-233. PGS_CSC_J2000toTOD Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_E_BAD_ARRAY_SIZE | incorrect array size |

**EXAMPLES:**

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double secTAI93 = -44496000.0;
PGSt_double posvelECI[6] = {0.5,0.75,0.90,0.3,0.2,0.8};
PGSt_double posvelTOD[6];

returnStatus=
PGS_CSC_J2000toTOD(6,secTAI93,posvelECI,posvelTOD);

if(returnStatus != PGS_S_SUCCESS)
{
/** test errors, take appropriate action **/
}
```

FORTRAN:

```
implicit none

integer           returnstatus
integer           pgs_csc_j2000totod
integer           threeor6
double precision  sectai93
```

```
double precision  posveleci(6)
double precision  posveltod(6)
integer           cnt1
character*33      err
character*241     msg

do 10 cnt1 = 1,6
      posveleci(cnt1) = 100 * cnt1
10 continue
sectai93 = -44496000.0
three0r6 = 6

returnstatus=s_csc_j2000totod(threeor6,sectai93,posveleci,
                                posveltod)

if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err, msg
endif
```

**NOTES:** If threeOr6 is 3, only position is transformed; if 6 then both position and velocity.

**REQUIREMENTS:** PGSTK - 0910, 1050

# Transform from ECI True of Date to ECI J2000 Coordinates

**NAME:** **PGS_CSC_TODtoJ2000( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_TODtoJ2000(
        PGSt_integer        threeOr6,
        PGSt_double         secTAI93,
        PGSt_double         posvelTOD[6],
        PGSt_double         posvelECI[6])
```

FORTRAN:

```
include 'PGS_CSC_4.f'

include 'PGS_SMF.f'

integer function pgs_csc_todtoj2000(threeor6,sectai93,posveltod,
                                    posveleci)
        integer
        double precision
        double precision posveltod(*)
        double precision posveleci(*)
```

**DESCRIPTION:** This function transforms from TOD (true of date) coordinates to ECI (J2000) coordinates.

**INPUTS:**

***Table 6-234. PGS_CSC_TODtoJ2000.c Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| threeOr6 | dimension of input vector | N/A | 3 | 6 |
| secTAI93 | TOD time posvel TOD[6] | seconds | | |
| posvelTOD[] | Vector (position and possibly velocity) in ECI TOD | | | |
| posvelTOD[0] | x position | meters | | |
| posvelTOD[1] | y position | meters | | |
| posvelTOD[2] | z position | meters | | |
| posvelTOD[3] | x velocity | m/s | | |
| posvelTOD[4] | y velocity | m/s | | |
| posvelTOD[5] | z velocity | m/s | | |

**OUTPUTS:**

*Table 6-235. PGS_CSC_TODtoJ2000.c Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| posvelECI[] | Vector (position and possibly velecity) in ECI J2000 | | | |
| posvelECI[0] | x position | meters | | |
| posvelECI[1] | y position | meters | | |
| posvelECI[2] | z position | meters | | |
| posvelECI[3] | x velocity | m/s | | |
| posvelECI[4] | y velocity | m/s | | |
| posvelECI[5] | z velocity | m/s | | |

**RETURNS:**

*Table 6-236. PGS_CSC_TODtoJ2000c Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_E_BAD_ARRAY_SIZE | incorrect array size |

**EXAMPLES:**

C:

```
PGSt_SMF_status   returnStatus
PGSt_double secTAI93 = -44496000.0
PGSt_double posvelTOD[6] = 0.5,0.75,0.90,0.3,0.2,0.8};
PGSt_double posvelECI[6];
returnStatus =
PGS_CSC_TODtoJ2000(6,secTAI93,posvelTOD,posvelECI);
if(returnStatus != PGS_S_SUCCESS
{
/** test errors, take appropriate  action **/
}
```

FORTRAN:

```
implicit none
integer           pgs_csc_todtoj2000
integer           returnstatus
integer           threeor6
double precision  sectai93
double precision  posveltod(6)
double precision  posveleci(6)
integer     cnt1
character*33      err
character*241     msg
```

```
do 10 cnt1  = 1,6
      posveltod(cnt1) = 100 * cnt1
10 continue
sectai93 = -44496000.0
three0r6 = 6
returnstatus=pgs_csc_todtoj2000(threeor6,sectai93,posveltod,
>                                   posveleci)
if (returnstatus .ne. pgs_s_success) then
      pgs_smf_getmsg(returnstatus, err, msg)
      write(*,*) err,
endif
```

**NOTES:** If threeor6 is 3, only position is transformed; if 6 then both position and velocity.

**TIME ACRONYMS:**

TAI is: International Atomic Time

TDB is: Barycentric Dynamical Time

TOOLKIT INTERNAL TIME (TAI):

Toolkit internal time is the real number of continuous SI seconds since the epoch of UTC 12 AM 1-1-1993. Toolkit internal time is also referred to inthe toolkit as TAI.

**REFERENCES FOR TIME:**

CCSDS 301.0-B-2 (CCSDS => Consultative Committee for Space Data Systems) Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac

**REQUIREMENTS:** PGSTK - 0910, 1050

# Determine if Location on Earth is in Day or Night

**NAME:** **PGS_CSC_DayNight( )**

**SYNOPSIS:**

C:

```
#include <PGS_CBP.h>

PGSt_SMF_status
PGS_CSC_DayNight(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         latitude[],
        PGSt_double         longitude[],
        PGSt_tag            sunZenithLimitTag,
        PGSt_boolean        afterDark[])
```

FORTRAN:

```
include 'PGS_MEM_7.f'
include 'PGS_CBP_6.f'
include 'PGS_CSC_4.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function pgs_csc_daynight(numvalues,asciiutc,offsets,latitude,
                                  longitude, sunzenithlimittag, afterdark)
        integer             numvalues
        character*27        asciiutc
        double precision    offsets(*)
        double precision    latitude(*)
        double precision    longitude(*)
        integer             sunzenithlimittag
        integer             afterdark(*)
```

**DESCRIPTION:** This function determines whether each point in a set of input Earth locations is in day or night at the corresponding input times. The function accepts an input start time, array of offsets from that start time, and an array of corresponding geodetic latitudes and longitudes. It then determines whether each time and point on the surface of the Earth (altitude = 0 km) is night, based on definitions of either civil twilight or night, nautical night, or astronomical night.

**INPUTS:**

***Table 6-237. PGS_CSC_DayNight Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of input time offsets, longitudes, and latitudes | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | 1972–01-01 | see NOTES |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC+ offset is between asciiUTC Min and Max values | |
| latitude | array of geodetic latitudes for array of time offsets | radians | -pi/2 | +pi/2 |
| longitude | array of longitudes corresponding to time offsets | radians | -2*pi | 2*pi |
| sunZenithLimitTag | tag specifying basis of day/night determination<br>Allowed values:<br>PGSd_CivilTwilight—(end of day) sun deemed to set within 90 degrees 50 arc minutes from zenith<br>PGSd_CivilNight—(end of civil twilight) sun more than 96 degrees from zenith (same as start of Nautical twilight)<br>PGSd_NauticalNight—(end of Nautical twilight) sun more than 102 degrees from zenith.<br>PGSd_AstronNight—(end of Astronomical Twilight) sun more than 108 degrees from zenith. | N/A | N/A | N/A |

**OUTPUTS:**

***Table 6-238. PGS_CSC_DayNight Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| afterDark | array of answers:<br>Array values will be either PGS_TRUE or PGS_FALSE, according to the tag definition. PGS_TRUE means point is in night, PGS_FALSE means point is in daylight or twilight. | Boolean | see DESCRIPTION | see DESCRIPTION |

**RETURNS:**

*Table 6-239. PGS_CSC_DayNight Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSTD_E_NO_LEAP_SECS | No leap second value available in table for at least one of the input offset times; a linear approximation was used to get value |
| PGSCSC_E_INVALID_LIMITTAG | Invalid sunZenithLimitTag |
| PGSCSC_E_BAD_ARRAY_SIZE | numValues (and array size) is less than zero |
| PGSCSC_W_ERROR_IN_DAYNIGHT | An error occurred in computing at least one afterDark value |
| PGSCSC_W_BAD_TRANSFORM_VALUE | Invalid ECItoECR transformation |
| PGSCSC_W_BELOW_HORIZON | Sun is below horizon |
| PGSCSC_W_PREDICTED_UT1 | At least one of the values obtained from the utcpole.dat file is 'predicted' |
| PGSTD_E_NO_UT1_VALUE | No UT1–UTC correction available |
| PGSTD_E_BAD_INITIAL_TIME | Initial input time cannot be deciphered |
| PGSCBP_E_TIME_OUT_OF_RANGE | Start UTC time is not in the range of the planetary ephemeris file (de200.eos) |
| PGSCBP_E_UNABLE_TO_OPEN_FILE | Ephemeris file cannot be opened |
| PGSMEM_E_NO_MEMORY | No memory available to allocate vectors |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES**:

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
PGSt_integer      counter;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE] =
                  {3600.0,7200.0,10800.0};
PGSt_double       latitude[ARRAY_SIZE]= {0.5,0.75,0.90};
PGSt_double       longitude[ARRAY_SIZE] = {1.0,2.0,3.0};
PGSt_tag          sunZenithLimitTag = PGSd_CivilTwilight;
PGSt_boolean      afterDark[ARRAY_SIZE];

numValues = ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CSC_DayNight(numValues,asciiUTC,offsets,
```

```
                                    latitude,longitude,
                                    sunZenithLimitTag,
                                    afterDark);

if(returnStatus != PGS_S_SUCCESS)
{
  ** test errors,
      take appropriate
       action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter <= numValues)
{
    printf("Offset: %lf   Latitude:%lf   Longitude:%lf
      Day/Night:%u", offset[counter],
            latitude[counter], longitude[counter],
            afterDark[counter]);
            counter++;
}
```

FORTRAN:

```
implicit none

integer           pgs_csc_daynight
parameter         (array_size=3)
integer           returnstatus
integer           counter
integer           numvalues
character*27      asciiutc
double precision  offsets(array_size)
double precision  latitude(array_size)
double precision  longitude(array_size)
integer           sunzenithlimittag
integer           afterdark(array_size)

data offsets/3600.0,7200.0,10800.0/
data latitude/0.5,0.75,0.90/
data longitude/1.0,2.0,3.0/
numvalues = array_size
asciiutc = '1991-01-01T11:29:30'
sunzenithlimittag = pgsd_civiltwilight

returnstatus = pgs_csc_daynight(numvalues,asciiutc,offsets,
                                    latitude,longitude,
                                    sunzenithlimittag,
                                    afterdark)
```

```
      if(returnstatus .ne. pgs_s_success) go to 90

      write(6,*) asciiutc
      if(numvalues.eq.0) numvalues = 1
      do 40 counter = 1,numvalues,1
         write(6,*)offsets(counter),latitude(counter),
                   longitude(counter),afterdark(counter)

   40 continue

   90 write(6,99)returnstatus

   99 format('ERROR:',I50)
```

**NOTES:** If there is an error in computing one or more of the afterDark values, which does not affect the computation of the other values for the input offset times, it is set to the returnStatus value.

An Earth model tag is not needed because the latitude is geodetic. Input latitude values should be based on an Earth model (flattening) consistent with that used for other data analysis and processing for the same spacecraft.

User supplies one of the four sunZenithLimitTags as part of the input information. Users wishing to know if a point is in Nautical Twilight or darker should use the Civil Night tag; those wishing to determine whether the point is after the start of Astronomical Twilight should use the Nautical Night tag. An example of tag usage is the following: if the tag is set to Nautical night and the Sun zenith angle is less than 102 degrees, afterDark will be false; if 102 degrees or more it will be true.

**TIME ACRONYMS:**

UTC is: Coordinated Universal Time

See Section 6.3.4.8 Coordinate System Conversion Tool Notes

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

**REQUIREMENTS:** PGSTK–0860, PGSTK–0930

# Calculate Nutation Angles

**NAME:** **PGS_CSC_wahr2( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>
PGS_CSC_wahr2(
        PGSt_double ddjd[2],
        PGSt_double dvnut[4])
```

FORTRAN:

```
include 'PGS_SMF.f'
integer function pgs_csc_wahr2(ddjd,dvnut)
        double precision    ddjd(2)
        double precision    dvnut(4)
```

**DESCRIPTION:** Calculates nutation angles delta psi and delta epsilon, and their rates of change, referred to the ecliptic of date, from the Wahr series.

**INPUTS:**

***Table 6-240. PGS_CSC_wahr2 Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| ddjd[2] | Barycentric Dynamical Time as a Julian Date | N/A | ANY | ANY |
| ddjd[0] | half-integral Julian day | | | |
| ddjd[1] | Julian day fraction | | | |

**OUTPUTS:**

***Table 6-241. PGS_CSC_wahr2 Outputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| dvnut[0] | nutation in longitude | radians | -0.01 | 0.01 |
| dvnut[1] | nutation in obliquity | radians | -0.001 | 0.001 |
| dvnut[2] | nutation rate in longitude | radians/sec | -1.16e-1 | +1.16e-11 |
| dvnut[3] | nutation rate in obliquity | radians/sec | -1.16e-13 | +1.16e-13 |

**RETURNS:**

***Table 6-242. PGS_CSC_wahr2 Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |

**EXAMPLES**:

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       jedTDB[2]={2449720.5,0.25};
PGSt_double       dvnut[4];

returnStatus = PGS_CSC_wahr2(jedTDB,dvnut);

** do something with shiny new nutation angles and rates **
                :
                :
```

FORTRAN:

```
implicit none

integer           pgs_csc_wahr2
integer           returnstatus
double precision  jedtdb(2)
double precision  dvnut(4)

data jedtdb/2449720.5,0.25/

returnstatus = pgs_csc_wahr2(jedtdb,dvnut)

!               do something with shiny new nutation angles and rates
                :
                :
```

**NOTES:**

From table 1, "proposal to the International Astronomical Union (IAU) working group on nutation," John M. Wahr and Martin L. Smith (1979) subroutine to compute nutation angles and rates from expressions given in Supplement to Astronomical Almanac 1984, S21–S26. Ref: P.K. Seidelmann, V.K. Abalakin, H. Kinoshita, J. Kovalevsky, C.A. Murray, M.L. Smith, R.O. Vicente, J.G. Williams, Ya. S. Yatskiv: 1982, "1980 IAU Theory of Nutation", Celestial Mechanics Journal, vol 27., p. 79–105

Changes to code prior to acquisition for ECS project:

Lieske 3/91. NUTATION in the IAU J2000 system. Univac version obtained from Myles Standish, (subroutine WAHR) who had obtained it from USNO. Re-ordered terms to match Astronomical Almanac 1984 table S23-S25 and corrected the rate for dPsi in the 0 0 2 -2 2 term. Eliminated the equivalencies, common block and added necessary SAVEs. Corrected the fundamental angles (L, L', F, D, Node) to match Almanac.

Acquired from E. Myles Standish, JPL, 12/93 by Peter Noerdlinger. This is not JPL certified code. Please do not modify the names of the variables in this code. It is heritage code and we may receive updates. We may also receive other related code with the same names for variables.

Users concerned with speed may wish to avoid repeated calls where possible. In this regard, the rates that are provided by Wahr2 can be used

either for estimating the error of using nearby times, or for short term extrapolation. Note that in the original JPL code the rates issued by wahr2 are in radians per day; this routine returns the rates as radians per second.

**REQUIREMENTS:** PGSTK–0916, PGSTK–0930, PGSTK–1050

# Get Greenwich Hour Angles

**NAME:** **PGS_CSC_GreenwichHour( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GreenwichHour(
        PGSt_integer        numValues,
        char                asciiUTC[28],
        PGSt_double         offsets[],
        PGSt_double         hourAngleGreenw[])
```

FORTRAN:

```
include 'PGS_CSC_4.f'
include 'PGS_TD_3.f'
include 'PGS_TD.f'
include 'PGS_SMF.f'

integer function
pgs_csc_greenwichhour(numvalues, asciiutc, offsets(*),
                      houranglegreenw(*))
        integer         numvalues,
        character*27        asciiutc,
        double precision    offsets(*),
        double precision    houranglegreenw(*)
```

**DESCRIPTION:** This function computes hour angle of the Vernal Equinox at the Greenwich meridian, accepting an input start time plus an array of time offsets.

**INPUTS:**

***Table 6-243. PGS_CSC_GreenwichHour Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| numValues | number of input time offsets | N/A | 0 | any |
| asciiUTC | UTC start time in CCSDS ASCII Time Code A or B format | N/A | see Notes | see Notes |
| offsets | array of time offsets | seconds | Max and Min such that asciiUTC + offset is between asciiUTC Min and Max values | |

**OUTPUTS:**

*Table 6-244. PGS_CSC_GreenwichHour Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| hourAngleGreenw | array of values of the hour angle of the Vernal Equinox at Greenwich; a value of 999999.0 is returned for invalid offset times | hours | 0 | 24 |

**RETURNS:**

*Table 6-245. PGS_CSC_GreenwichHour Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_ERRORS_IN_GHA | An error occurred in computing at least one Greenwich hour angle |
| PGSCSC_W_PREDICTED_UT1 | Data in utcpole.dat file is predicted (not final) value for at least one input time |
| PGSTD_E_TIME_VALUE_ERROR | Error in input time value |
| PGSTD_E_TIME_FMT_ERROR | Error in input time format |
| PGSTD_E_NO_LEAP_SECS | No leap seconds correction is available in leapsec.dat file for at least one of the input times/offsets |
| PGS_E_TOOLKIT | Something unexpected happened, execution of function ended prematurely |

**EXAMPLES:**

C:

```
#define     ARRAY_SIZE  3

PGSt_SMF_status   returnStatus;
PGSt_integer      numValues;
PGSt_integer      counter;
char              asciiUTC[28];
PGSt_double       offsets[ARRAY_SIZE]=
                  {3600.0,7200.0,10800.0};
PGSt_double       hourAngleGreenw[ARRAY_SIZE];

numValues=ARRAY_SIZE;
strcpy(asciiUTC,"1991-01-01T11:29:30");
returnStatus = PGS_CSC_GreenwichHour(numValues,asciiUTC,
                                     offsets,
                                     hourAngleGreenw);
```

```
if(returnStatus != PGS_S_SUCCESS)
{
** test errors,
   take appropriate
   action **
}
printf("start time:%s",asciiUTC);
counter = 0;
while(counter < numValues)
{
    printf("Offset: %lf   Hour Angle:%lf",offset[counter],
        hourAngleGreenw[counter]);
    counter++;
}
```

FORTRAN:

```
      implicit none

      parameter (array_size=3)
      integer           pgs_csc_greenwichhour
      integer           array_size
      integer           returnstatus
      integer           counter
      integer           numvalues
      character*27      asciiutc
      double precision  offsets(array_size)
      double precision  houranglegreenw(array_size)

      data offsets/3600.0,7200.0,10800.0/
      array_size = 3
      numvalues = array_size
      asciiutc = '1991-01-01T11:29:30'

      returnstatus = pgs_csc_greenwichhour(numvalues,asciiutc,
                                           offsets,
                                           houranglegreenw)

      if(returnstatus .ne. pgs_s_success) go to 90
      write(6,*) asciiutc
      if(numvalues.eq.0) numvalues = 1
      do 40 counter = 1, numvalues,1
      write(6,*)offsets(counter),houranglegreenw(counter)

   40 continue

   90 write(6,99)returnstatus

   99 format('ERROR:',A50)
```

**NOTES**: Historically, UT1 was used as a measure of time, but since 1958 it has served only as a measure of Earth rotation. The only real difference between UT1 and Greenwich Mean Sidereal Time (GMST) is that UT1 measures Earth rotation in regards to the vector from Earth center to the mean sun (a fictitious point that traverses the celestial equator at the same mean rate that the sun apparently traverses the ecliptic), while GMST measures Earth rotation relative to the vernal equinox. Essentially, the value of GMST in radians is larger than that of UT1 in radians by the ratio of the mean solar day to the sidereal day; however, there are small correction terms due to precession. The equation used in function PGS_TD_gmst( ) is valid for the period 1950 to well past 2000, as long as the definition of UT1 and the reference equinox (J2000) are not changed. The basic limitation is the accuracy of UT1. Users obtaining UT1 from the SDP Toolkit should observe time limitations in the function PGS_TD_UTCtoUT1( ).

**TIME ACRONYMS:**

GMST is: Greenwich Mean Sidereal Time
TAI is: International Atomic Time
UT1 is: Universal Time
UTC is: Coordinated Universal Time

See Section 6.2.7.5.2 (UT1-UTC Boundaries)

**REFERENCE FOR TIME:**

CCSDS 301.0–B–2 (CCSDS => Consultative Committee for Space Data Systems Astronomical Almanac, Explanatory Supplement to the Astronomical Almanac.

**REQUIREMENTS:** PGSTK–0770

# Get Zenith and Azimuth of an ECR Vector at the Look Point

**NAME:** **PGS_CSC_ZenithAzimuth( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_ZenithAzimuth(
        PGSt_double     vectorECR[3],
        PGSt_double     latitude,
        PGSt_double     longitude,
        PGSt_double     altitude,
        PGSt_tag        vectorTag,
        PGSt_boolean    zenithOnlyFlag,
        PGSt_boolean    refractFlag,
        PGSt_double     *zenith,
        PGSt_double     *azimuth
        PGSt_double     *refraction)
```

FORTRAN:

```
include'PGS_CSC.f'
include'PGS_CSC_4.f'
include'PGS_SMF.f'

integer function
pgs_csc_zenithazimuth(vectorecr,latitude,longitude,altitude,
                                        vectortag,zenithonlyflag,
                                        refractflag,zenith,azimuth,
                                        refraction)
        double precision    vectorECR[3]
        double precision    latitude
        double precision    longitude
        double precision    altitude
        integer             vectortag
        integer             zenithonlyflag
        integer             refractflag
        double precision    zenith
        double precision    azimuth
        double precision    refraction
```

**DESCRIPTION**: Computes the zenith and the azimuth of a vector at the look point. This tool allows for refraction if desired.

**INPUTS:**

*Table 6-246. PGS_CSC_ZenithAzimuth Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| vectorECR | ECR vector whose zenith & azimuth is desired (in case of PGSD_MOON do not use a unit vector!—see NOTES) | meters or unit vector | N/A | N/A |
| latitude | geodetic latitude | radian | -pi/2 | pi/2 |
| longitude | longitude | radian | -2*pi | 2*pi |
| altitude | altitude off the geoid (altitude is an input only when refracFlag is PGS_TRUE see NOTES). | meters | -2000 | 80000 |
| vectorTag | PGSd_CB, PGSd_moon, or PGSd_Look or a CB identifier (see NOTES) | N/A | N/A | N/A |
| zenithOnlyFlag | omit azimuth calculation | N/A | N/A | N/A |
| refracFlag | turns on refraction | N/A | N/A | N/A |

**OUTPUTS:**

*Table 6-247. PGS_CSC_ZenithAzimuth Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| zenith | zenith angle | radian | 0 | 1.6755 (96 deg) |
| azimuth | azimuth E from N | radian | -pi | +pi |
| refraction | increase of zenith angle due to refraction | radian | 0 | 0.1 |

**RETURNS:**

*Table 6-248. PGS_CSC_ZenithAzimuth Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful execution |
| PGSCSC_W_BELOW_HORIZON | Warning indicating the object is below horizon |
| PGSCSC_W_UNDEFINED_AZIMUTH | The object is at the zenith. In this case azimuth is not calculated |
| PGSCSC_W_NO_REFRACTION | No refraction calculation done due to errors |
| PGSCSC_E_INVALID_VECTAG | The input vector tag is not PGSd_CB, PGSd_MOON, PGSd_LOOK or a celestial body identifier |
| PGSCSC_E_LOOK_PT_ALTIT_RANGE | Look point altitude not reasonable |
| PGSCSC_E_ZERO_INPUT_VECTOR | The input vector has zero length |
| PGS_E_TOOLKIT | Unknown error occurred |

**EXAMPLES:**

C:

```
PGSt_SMF_status   returnStatus;
PGSt_double       vectorECR[3];
PGSt_double       latitude  = 0.2;
PGSt_double       longitude = 0.1;
PGSt_double       altitude = 0.0;
PGSt_tag          vectorTag = PGSd_LOOK;
PGSt_boolean      zenithOnlyFlag = PGS_FALSE;
PGSt_boolean      refractFlag = PGS_FALSE;
PGSt_double       zenith;
PGSt_double       azimuth;
PGSt_double       refraction;

vectorECR[2] = -0.26;
vectorECR[1] = 0.0;
vectorECR[0] = sqrt(1 - vectorECR[2]*vectorECR[2]);

returnStatus = PGS_CSC_ZenithAzimuth(vectorECR,latitude,
                                     longitude,altitude,
                                     vectorTag,
                                     zenithOnlyFlag,
                                     refractFlag,&zenith,
                                     &azimuth,&refraction)

 do some error handling
 if desired, convert zenith and azimuth to degrees
printf("zenith angle = %lf,  azimuth = %lf\n", zenith,
        azimuth);
```

FORTRAN:

```
implicit none

integer           pgs_csc_zenithazimuth
double precision  look[3]
double precision  latitude
double precision  longitude
double precision  altitude
integer           zenithonlyflag
integer           refractflag
double precision  zenith
double precision  azimuth
double precision  refraction
integer           vectortag,returnstatus

vectortag = pgsd_look
latitude  = 0.2D0
longitude = -0.3D0
```

```
                    altitude  = 0.0D0
                    zenithonlyflag = PGS_FALSE
                    refractflag    = PGS_FALSE

                    look[3] = -0.26;
                    look[2] = 0.0;
                    look[1] = sqrt(1 - look[3]*look[3]);

                    returnstatus = pgs_csc_zenithazimuth(look,latitude,
                                                         longitude,altitude,
                                                         vectortag,
                                                         zenithonlyflag,
                                                         refractflag,zenith,
                                                         azimuth,refraction)

C       do some error handling

C       if desired, convert zenith and azimuth to degrees
```

**NOTES**: The vectorECR vector must be in ECR coordinates. For celestial bodies, it is the vector from Earth to the celestial body. It can be obtained by getting the ECI vector to the body from PGS_CBP_Earth_CB_Vector(), and transforming that vector to ECR rectangular coordinates with PGS_CSC_ECItoECR().

The "look vector" (which could as well be called the "boresight vector") is an ECR vector from the instrument to the point being viewed )"look point."). To obtain the zenith and azimuth of the look vector, the vectorTag must be set to PGSd_LOOK (this allows for the reversed sense of such a vector, which represents a line of sight above the horizon when pointing down). If desired, the unit look vector can be obtained and saved from PGS_CSC_GetFOV_Pixel( ); this will achieve very good performance, as the ECR look vector is calculated there. If the ECR look vector has to be constructed (for example, when starting with already-geolocated data), this can be done, for example, as follows: Convert the latitude, longitude and altitude of the look point to ECR rectangular coordinates with PGS_CSC_GEOtoECR(). Then obtain the ECI spacecraft position from PGS_CSC_Ephem_Attit(), and convert it to ECR with PGS_CSC_ECItoECR(). Finally, subtract the last result from the first. The geometry is illustrated in Fig 6-4:

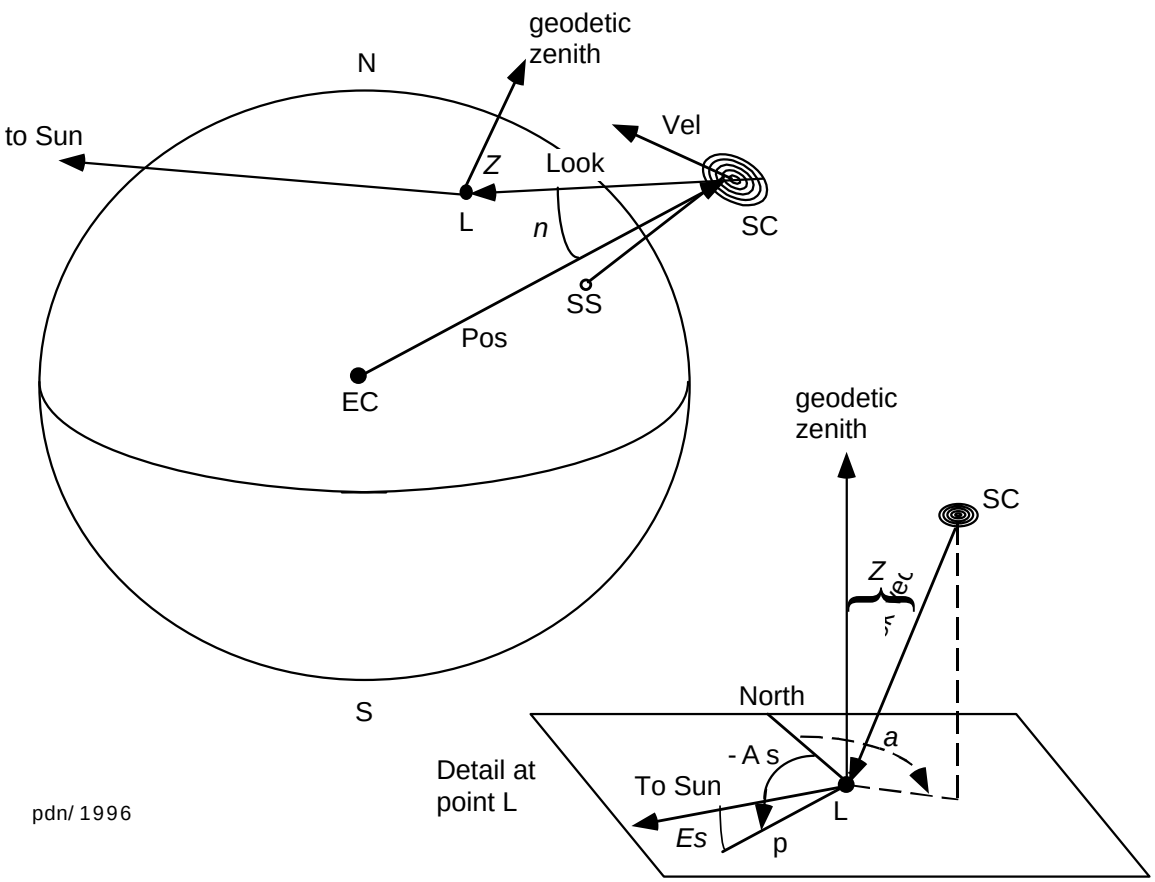



***Figure 6-4. Geometry of the Viewing and Sun Vectors***

Notation:
SC = spacecraft
EC = Earth Center
Pos = position vector
Vel = velocity vector
SS = subsatellite point
L = Lookpoint
*n* = nadir angle
*Es* = elevation of the Sun = ($\pi/2$ - solar zenith angle).
p = projection of Sun vector on horizontal
As = solar azimuth. All azimuths are measured East from North
*Z* = zenith angle of look vector. *a* = azimuth of the Look vector.

Note that because of Earth curvature, generally $Z > n$. Also, in this diagram m,*n* is referenced to geocentric, SS to geodetic, a small difference. In other contexts, however, "nadir angle" could refer to the

angle between a direction and either geocentric, geodetic, or nominal spacecraft nadir.

Note: In ECR coordinates, the Look Vector is also called the ECR Pixel Vector, but in SC coordinates, it is called the SC Pixel Vector.

If the zenith and azimuth of a distant celestial body (such as the sun or a planet) are desired, the user may supply PGSd_CB or any of the identifiers: PGSd_SUN, PGSd_MERCURY, PGSd_VENUS, PGSd_MARS, PGSd_JUPITER, PGSd_SATURN, PGSd_URANUS, PGSd_NEPTUNE, or PGSd_PLUTO. This is purely a convenience for users doing other calculations with a CB identifier; the action of the function is in all cases the same—it finds the zenith and azimuth of the vector at the look point, without regard to parallax (i.e., the vector from Earth center to the Celestial body is regarded as unchanged due to the displacement of the look point from Earth center).

In the case of the PGSd_MOON, the geocentric parallax is appreciable, meaning that its apparent position is, in general, different as viewed from Earth center or from the look point. The difference can be as large as a degree. Therefore, in this case, a parallax correction is made. It is essential, in this case, of course, that the PGSd_MOON vector be supplied in meters. In this case, the input vector should be the Earth to PGSd_MOON vector defined from Earth center (geocentric), as obtained, for example, from the PGS_CBP_Earth_CB_Vector( ) tool.

In all other cases, the input vector can be in any units, including normalized (unit vector).

Users wishing to take into account the minuscule parallax correction for the sun, or the correction for some other chosen body such as an asteroid, could simply label the vector as PGSd_MOON. (For the sun, the correction is only ~ 2.5 millidegrees.)

Refraction by the atmosphere is calculated if the flag is set to PGS_TRUE. This calculation approximately corrects, in the visual band, for the fact that any line of sight, such as the sun, moon, or look vector is bent by the atmosphere.

If the vector is well below the horizon, a warning is returned and no azimuth calculation is done. The present algorithm is fairly forgiving for points slightly below the horizon (to 96 degrees), in order that the user interested in the location of the glow before sunrise or after sunset can find its azimuth; it is user responsibility to take special action between 90 degrees and 96 degrees if these data are not wanted.

The altitude is required only if refraction is to be calculated, and its only effect is to change the mean density of the atmosphere in the refraction function.

If the zenith only flag is defined by the user to be PGS_TRUE the function will run faster but will not calculate the azimuth.

If the azimuth is requested but the zenith angle is < 0.026 deg, it is deemed that the azimuth calculation is unreliable, because variations in the local vertical as determined from the geoid, and variable refraction in the atmosphere dominate at that level. The azimuth is returned as 0.0 and the warning PGSCSC_W_UNDEFINED_AZIMUTH is returned

The calculation herein is entirely independent of the Earth model except for the parallax correction, where WGS84 is assumed, and any difference in other models introduces negligible error. The use of geodetic latitude as input guarantees that the rest of the algorithm is independent of Earth model.

**REQUIREMENTS:** PGSTK–1091

# Find Point of Closest Miss and Surface Point

**NAME:** **PGS_CSC_GrazingRay( )**

**SYNOPSIS:**

C:

```
#include <PGS_CSC.h>

PGSt_SMF_status
PGS_CSC_GrazingRay(
        char            earthEllipsTag[50],
        PGSt_double     posECR[3],
        PGSt_double     ray[3],
        PGSt_double     *latitude,
        PGSt_double     *longitude,
        PGSt_double     *missAltitude,
        PGSt_double     *slantRange,
        PGSt_double     posNEAR[3],
        PGSt_double     posSURF[3])
```

FORTRAN:

```
include 'PGS_SMF.f'
include 'PGS_CSC_4.f'

integer function pgs_csc_grazingray(
        earthellipstag,pos,ray,latitude,longitude,
        missaltitude,slantrange,posnear,possurf)

character*49        earthellipstag
double precision    posecr(3)
double precision    ray(3)
double precision    latitude
double precision    longitude
double precision    missaltitude
double precision    slantrange
double precision    posnear(3)
double precision    possurf(3)
```

**DESCRIPTION:** For rays that miss Earth limb, this function finds the nearest miss point on the ray and the corresponding surface point. For rays that strike the Earth, it finds instead the midpoint of the chord of the ray within the ellipsoid and the surface point of intersection nearest the observer.

**INPUTS:**

*Table 6-249. PGS_CSC_GrazingRay Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| earthEllipsTag | tag selecting Earth ellipsoid model (default is WGS84) | N/A | N/A | N/A |
| posECR | ECR Spacecraft Position | meters | N/A | see NOTES |
| ray[3] | unit vector along the line of sight, in ECR coordinates | N/A | -1 per component | +1 per component |

**OUTPUTS:**

*Table 6-250. PGS_CSC_GrazingRay Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| latitude | geodetic latitude of posNEAR (q.v. below) | radians | -pi/2 | pi/2 |
| longitude | longitude of posNEAR (q.v. below) | radians | -pi | pi |
| missAltitude | altitude of posNEAR (q.v. below) | meters | N/A | N/A |
| slantRange | range to posNEAR (q.v. below) | m/s | -7000 | 7000 |
| posNEAR[0] | X coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES) | meters | N/A | N/A |
| posNEAR[1] | Y coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES) | meters | N/A | N/A |
| posNEAR[2] | Z coordinate (in ECR) of the point on ray nearest to the ellipsoid (when ray misses); when ray hits, this point is defined to be midpoint of the ray chord within the ellipsoid (see NOTES) | meters | N/A | N/A |
| posSURF[0] | X coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid, (in the case that it does not miss) | meters | N/A, but normally < 6378140 | N/A, but normally > -6378140 |
| posSURF[1] | Y coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid (in the case that it does not miss) | meters | N/A, but normally < 6378140 | N/A, but normally > -6378140 |
| posSURF[2] | Z coordinate (in ECR) of the point on Earth closest to the ray (when the ray misses Earth limb) or where the ray first strikes the Earth ellipsoid (in the case that it does not miss) | meters | N/A, but normally < 6378140 | N/A, but normally > -6378140 |

**RETURNS:**

***Table 6-251. PGS_CSC_GrazingRay Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCSC_W_SUBTERRANEAN | User provided a subterranean position for the spacecraft |
| PGSCSC_W_HIT_EARTH | Line of Sight struck the Ellipsoid |
| PGSCSC_W_LOOK_AWAY | Line of sight points away from Earth |
| PGSCSC_W_ERROR_IN_GRAZINGRAY | Generic return for warning in lower level function |
| PGSCSC_W_SPHERE_BODY | Using a spherical Earth model |
| PGSCSC_W_LARGE_FLATTENING | Issued if flattening factor is greater than 0.01 |
| PGSCSC_W_DEFAULT_EARTH_MODEL | Default Earth model was used - user's model not found |
| PGSCSC_W_ZERO_PIXEL_VECTOR | Zero length ray vector supplied (terminates execution) |
| PGSCSC_E_BAD_EARTH_MODEL | Equatorial or polar radius less than or equal to 0.0 or the model defines a prolate Earth |
| PGS_E_TOOLKIT | Something unexpected happened—execution aborted |

**EXAMPLES**:[1]

```
C:          #include <PGS_CSC.h>
            PGSt_SMF_status   returnStatus
                  char              earthEllipsTag[50];
                  PGSt_double       posECR[3];
                  PGSt_double       ray[3];
                  PGSt_double       latitude;
                  PGSt_double       longitude;
                  PGSt_double       missAltitude;
                  PGSt_double       slantRange;
                  PGSt_double       posNEAR[3];
                  PGSt_double       posSURF[3];

            strcpy(earthEllipsTag,"GEM-10B");
            posECR[0] = 4077000.0;
                  posECR[1] = 5000000.0;
                  posECR[2] = -3200000.0;
            ray[0] = 0.0002;
                  ray[1] = -1.0;
                  ray[2] = -0.422;
            returnStatus = PGS_CSC_GrazingRay(
                  earthEllipsTag,posECR,ray,&latitude,&longitude,
                  &missAltitude,&slantRange, posNEAR,
                  posSURF);
```

[1] Note: As is Toolkit standard, to avoid possible interface problems to outside software that may support different word lengths, the Toolkit renormalizes all unit input vectors at the same time it checks for zero length input vectors. Therefore the inputs shown here are unnormalized.

```
            printf("Longitude %f\n",longitude);
                  printf("Latitude: %f\n",latitude);
                  printf("Altitude: %f\n",missAltitude);
                  printf("Slant Range: %f\n",slantRange);
            if(returnStatus == PGS_S_SUCCESS)
                  {
                    printf("Point on Ray Nearest Earth: %f, %f, %f\n",
                    posNEAR[0],posNEAR[1],posNEAR[2]);
                    printf("Point on Surface Nearest Ray: %f, %f, %f\n",
                    posSURF[0],posSURF[1],posSURF[2]);
                  }
            else if(returnStatus == PGSCSC_W_HIT_EARTH)
                  {
                  printf("Midpoint of Ray Chord in Earth: %f, %f, %f\n",
                     posNEAR[0],posNEAR[1],posNEAR[2]);
                  printf("Line of Sight Strikes Earth at: %f, %f, %f\n",
                              posSURF[0],posSURF[1],posSURF[2]);
                  }
            else
                  {
              ** test errors,
                  take appropriate
                   action **
                  }
```

FORTRAN:

```
            include 'PGS_SMF.f'
            include 'PGS_CSC_4.f'
            implicit none
            integer           pgs_csc_grazingray
            integer           returnstatus
            character*19      earthellipstag
            double precision  posecr(3)
            double precision  ray(3)
            double precision  latitude
            double precision  longitude
            double precision  missaltitude
            double precision  slantrange
            double precision  posnear(3)
            double precision  possurf(3)

            posecr(1) = 4077000.0
            posecr(2) = 5000000.0
            posecr(3) = -3200000.0
            ray(1) = -0.0002
            ray(2) = -1.0
            ray(3) = -0.422
            returnstatus = pgs_csc_grazingray(
            earthellipstag,posecr,ray,latitude,longitude,
                  missaltitude,slantrange, posnear,
                  possurf)
            print*,'Longitude: ',longitude
            print*,'Latitude: ',latitude
            print*,'Slant Range: ',slantrange
            print*,'Altitude: ',missaltitude

      if(returnStatus .eq. PGS_S_SUCCESS) then
C ampersands & below are continuation marks in column
```

```
         print*,'Point on Ray Nearest Earth: X = ',
         posnear(1),'  Y = ',posnear(2),' Z = ', posnear(3)
         print*,'Point on Surface Nearest Ray: X = ',
     &   possurf(1),'  Y = ',possurf(2),' Z = ', possurf(3)
      else if (returnStatus .eq. PGSCSC_W_HIT_EARTH) then
         print*,'Midpoint of Ray Chord in Earth: X = ',
     &posnear(1),'  Y = ',posnear(2),' Z = ', posnear(3)
         print*,'Line of Sight Strikes Earth at: X = ',
     &possurf(1),'  Y = ',possurf(2),' Z = ', possurf(3)
      else
C  ** test errors, take appropriate action **
      endif
      print*,err,msg
```

**NOTES:** For a line of sight ("ray") that misses Earth limb, this tool calculates the rectangular coordinates of the point Q of closest approach to the Earth and the slant range to Q. It also obtains the latitude and longitude of the surface point P nearest Q (and therefore nearest to the ray) and the geodetic altitude of Q above P (Q and P have the same longitude and geodetic altitude). When the ray, instead, intersects the Earth ellipsoid, the rectangular coordinates of Q are replaced by those of a point halfway between the two "pierce points" where the ray intersects the ellipsoid. The intent is to provide a point with the nearly the same latitude and longitude as the point closest to the Earth's surface on a ray from a background object (such as the Sun) that is actually refracted round the Earth. When the ray intersects the Earth, the latitude and longitude of P are also replaced by those of the nearest pierce point, i.e. where the instrument is looking, and the slant range is replaced by the range to that point. Furthermore, the return value PGSCSC_W_HIT_EARTH issues. If the ray, instead, points *away* from the Earth ellipsoid, the altitude output variable is set to PGSd_GEO_ERROR_VALUE and the return value to PGSCSC_W_LOOK_AWAY, in either case - ray missing the ellipsoid or ray striking it. The return PGSCSC_W_ZERO_PIXEL_VECTOR terminates execution, even though it is a "warning" level only; the "warning" status was defined for this message to support certain tools that process arrays of pixels at once, in order that if a few pixel vectors were bad, the remainder could be processed. The same return is reused here, but promoted to have a fatal result.

If an invalid earthEllipsTag is input, the program will use the WGS84 Earth model by default.

**REQUIREMENTS:** PGSTK-1085

### 6.3.4.11 CSC Functions

**PGS_CSC_DayNight**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ECItoECR**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ECItoORB**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ECItoSC**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ECRtoECI**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ECRtoGEO**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_EarthOccult**

Test for earth occultation of a celestial body in the field of view. The test is in three phases. The first phase does not depend on the CB at all - it is just a check if the Earth fills the field of view. The second test (exercised only if the first fails to find total occultation) determines if the celestial body is behind the Earth. If the second test fails, the vector in SC coordinates that points at the part of the CB most distant from the Earth center is returned so that the calling function can determine if the Earth's bulge (difference in radius over that of an inscribed sphere) occults the CB.

**PGS_CSC_Earthpt_FixedFOV**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_Earthpt_FOV**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_EulerToQuat**

Transforms Euler angles to Quaternions.

**PGS_CSC_FOVconicalHull**

A circular cone is drawn around the FOV and a check is made as to whether the candidate point is inside it before going any further. The function has two purposes:

a. it will speed up tasks by obviating complicated algorithms for points well away from the FOV

b. it will enable detection and rejection of FOV specifications outside our present algorithmic limits. [Present software does not reliably handle fields of view more than 180 degrees across.]

**PGS_CSC_GEOtoECR**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_GetEarthFigure**

This tool gets the equatorial and polar radii from the earthfigure.dat file for the earth model input.

**PGS_CSC_GetFOV_Pixel**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_GreenwichHour**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_J2000toTOD**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_LookPoint**

Solves the look point equation. method:

Solve the quadratic equation

x = p + d*u

where x must lie on an ellipsoid, for the slant range, d, corresponding to the intersection of the extended look vector, u, with the surface of the earth. Then compute x directly.

**PGS_CSC_Norm**

This tool computes the norm of a 3-vector.

**PGS_CSC_ORBtoECI**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_ORBtoSC**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_PointInFOVgeom**

For each input point, the function does the processing to determine if a point is in the field of view and returns a flag indicating whether the point is in the field of view.

**PGS_CSC_QuatToEuler**

This function gets Euler angles from a quaternion.

**PGS_CSC_SCtoECI**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_SCtoORB**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_SpaceRefract**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_SubSatPoint**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_SubSatPointVel**

This tool finds the North and East components of the velocity of the subsatellite point and the rate of change of spacecraft altitude.

**PGS_CSC_TODtoJ2000**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_TiltYaw**

Obtains the tipped orbital (geodetic nadir) to orbital transformation quaternion.

**PGS_CSC_UT1_update**

updates the file “utcpole.dat”

**PGS_CSC_UTC_UT1Pole**

This tool accesses the file 'utcpole.dat' and extracts using interpolation the x,y pole postion in seconds of arc and the difference of UT1 and UTC, given an input Julian date.

**PGS_CSC_ZenithAzimuth**

See description in 6.3.4.9: Coordinate System Conversion Transformation Tools

**PGS_CSC_crossProduct**

Cross product of vectors

**PGS_CSC_dotProduct**

Dot product of vectors

**PGS_CSC_getECItoORBquat**

This function returns a quaternion describing the rotation from the Earth Centered Inertial (ECI) reference frame to the Orbital (ORB) reference frame. That is the quaternion returned will transform a vector in the ECI frame to the equivalent vector in the ORB frame.

**PGS_CSC_getORBtoECIquat**

This function returns a quaternion describing the rotation from the Orbital (ORB) reference frame to the Earth Centered Inertial (ECI) reference frame. That is the quaternion returned will transform a vector in the ORB frame to the equivalent vector in the ECI frame.

**PGS_CSC_getQuats**

Converts a transformation matrix to a quaternion

**PGS_CSC_nutate2000**

This tool transforms a vector under nutation between Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB) and True Celestial Coordinates of date.

**PGS_CSC_precs2000**

This tool precesses a vector from Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB) to J2000 coordinates or from J2000 coordinates to Mean Celestial Coordinates of date in Barycentric Dynamical Time (TDB).

**PGS_CSC_quatMultiply**

This file contains the function PGS_CSC_quatMultiply(). This function multiplies two quaternions, using a short algorithm with 11 multiplications and 19 additions.

**PGS_CSC_quatRotate**

This function transforms a vector from one coordinate system to a rotated coordinate system with a common origin, where the rotation is defined by a quaternion.

**PGS_CSC_quickWahr**

Wahr nutation with extrapolation for up to 1/2 hour (valid to microseconds of arc)

**PGS_CSC_wahr2**

Calculates nutation angles delta psi and delta epsilon, and their rates of change, referred to the ecliptic of date, from the Wahr series.

### 6.3.5 Geo–Coordinate Transformation Tools

The geo–coordinate transformation tools are required to support the bi–directional transformation between geographic coordinates and various standard map projection frames. The tools are designed to give rapid coordinate transformations for many points. The tool provides transformations from geodetic latitude and longitude as output from the geolocation tool into the required map projection frames.

There are initialization routines for each set of transformations (PGS_GCT_Init). PGS_GCT_Init routine is used to “save” in memory the projection parameters and to pre–calculate any variables that are required in all subsequent transformations. Following the initialization of a projection

there are routines for the forward and reverse transformations combined into a single interface (PGS_GCT_Proj).

The tool may be used to perform many transformations in different projections within the same executable program. If the same projection is required twice to perform transformations with two different sets of parameters, then the initialization routine has to be called before each set of transformations.

Every effort should be made to standardize projection definitions throughout the project to enable data sets to be generated in a consistent manner.

In addition to the standard transformations of latitude and longitude to map projection specific transformations from one projection to another are required, these are treated in a similar manner as discussed below.

The tool is based on the commonly available packages general cartographic transformation package (GCTP) for coordinate transformation; this package is based on the projections described by Snyder. *Map Projections—A Working Manual*—J.P. Snyder, USGS professional paper 1395, 1987.

# Initialize Given Projection Parameters

**NAME:** **PGS_GCT_Init( )**

**SYNOPSIS:**

C:

```
#include <PGS_GCT.h>

PGSt_SMF_status
PGS_GCT_Init(
        PGSt_integer        projId,
        PGSt_double         projParam[],
        PGSt_integer        directFlag)
```

FORTRAN:

```
include"PGS_GCT.f"
include"PGS_GCT_12.f"
include"PGS_SMF.f"

integer function pgs_gct_init( projid, projparam, directflag)
        integer          projid
        double precision(30)   projparam
        integer          directflag
```

**DESCRIPTION:** This tool provides a general interface to perform geo–coordinate transformations in the forward/inverse directions. In general the tool requires a projection id, location of input data vectors and the direction of the conversion. PGSd_UTM projection is a special case for which zone value is also needed to define a point.

**INPUTS:**

***Table 6-252. PGS_GCT_Init Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| projId | projection code | none | 1 | #defined |
| projParam | projection parms | rad, m if latitude<br>if longitude | -90(PI/180)<br>-PI | 90(PI/180)<br>PI |
| directFlag | forward/inverse | none | PGSd_GCT_FORWARD | PGSd_GCT_INVERSE |

**OUTPUTS**: None

**RETURNS**:

***Table 6-253. PGS_GCT_Init Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | |
| PGSGCT_E_NO_DATA_FILES | Data files for state plane could not be found |
| PGSGCT_E_GCTP_ERROR | Error has occurred in the GCTP lib |
| PGSGCT_E_BAD_INC_ANGLE | Invalid inclination angle in the Space Oblique Mercator (SOM) projection |
| PGSGCT_E_BAD_RADIUS | Invalid radius |
| PGSGCT_E_BAD_MINOR_AXIS | Invalid minor radius |
| PGSGCT_E_MINOR_GT_MAJOR | Minor radius is greater than major radius |
| PGSGCT_E_BAD_LONGITUDE | Invalid longitude |
| PGSGCT_E_BAD_LATITUDE | Invalid latitude |
| PGSGCT_E_BAD_DIRECTION | Invalid direction |
| PGSGCT_E_INVD_SPCS_SPHEROID | Invalid State Plane Coordinates Spheroid (SPCS) |
| PGSGCT_E_INVD_PROJECTION | Invalid Projection |

**EXAMPLES:** NONE (see example for PGS_GCT_Proj( ))

**NOTES:** This routine simply initializes the parameters required by a particular projection. The user is referred to the following appendices for further details

Projection List—Appendix G
Parameter List and Use—Appendix G
Spheroid List—Appendix G (State Plane Projection only)

Following steps should be taken if a new projection is to be added to the projection library:

Step 1—archive new code to the projection library

Step 2—define projection code for the new projection in proj.h

Step 3—increment the value of MAXPROJ by one in proj.h

Step 4—add calls to the forward and inverse initialization routines at the end of this file.

Parameters 0 and 1 are reserved for major axis and minor axis respectively

Parameter 4 is reserved for longitude values only.

Parameter 5 is reserved for latitude values only.

Parameters 6 and 7 are reserved for false easting and northing values only.

IMPORTANT All blank array elements are set to zero by the user

Latitude and longitude ranges are as defined in the input section above. The routine checks the longitude value as -PI <= longitude <= PI and latitude as -PI/2 <= latitude <=PI/2. The value of PI is defined as 3.141592653589793238 which is available to the user

**State Plane Projection is not available in the Toolkit.**

**REQUIREMENTS:** PGSTK–1500, PGSTK–1502

## Transforms Geographical Coordinates into Cartesian Coordinates and Vice Versa for the Given Projection

**NAME:** **PGS_GCT_Proj( )**

**SYNOPSIS:**

C:

```
#include <PGS_GCT.h>

PGSt_SMF_status PGS_GCT_Proj(
        PGSt_integer        projId,
        PGSt_integer        directFlag,
        PGSt_integer        nPoints,
        PGSt_double         longitude[],
        PGSt_double         latitude[],
        PGSt_double         mapX[],
        PGSt_double         mapY[],
        PGSt_integer        zone[]);
```

FORTRAN:

```
include"PGS_GCT.f"
include"PGS_GCT_12.f"

integer function pgs_gct_init( projid, directflag, npoints, longitude,
                                       latitude, mapx, mapy, zone)
        integer             projid
        integer             directflag
        integer             npoints
        double precision    longitude(*)
        double precision    latitude(*)
        double precision    mapx(*)
        double precision    mapy(*)
        double precision    zone(*)
```

**DESCRIPTION:** This tool provides a general interface to perform geo–coordinate transformations in the forward/inverse directions. In general the tool requires a projection id, location of input data vectors and the direction of the conversion. PGSd_UTM projection is a special case for which zone value is also needed for inverse transformations. Forward PGSd_UTM transformations return zone values as output.

**INPUTS:**

*Table 6-254. PGS_GCT_Proj Inputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| projId | projection code | none | 1 | #defined |
| directFlag | forward/inverse | none | PGSd_GCT_FORWARD | PGSd_GCT_INVERSE |
| nPoints | num. of points | none | 1 | variable |
| longitude[] | longitude values | radians | -PI | +PI |
| latitude[] | latitude values | radians | -PI | +PI |
| mapX[] | x cartesian coordinate (see notes) | meters | variable | variable |
| mapY[] | y cartesian coordinate (see notes) | meters | variable | variable |
| zone[] | UTM zones (negative for southern hemisphere) | none | -60 | 60 |

**OUTPUTS**: See description

**RETURNS**:

*Table 6-255. PGS_GCT_Proj Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSGCT_E_BAD_ZONE | Invalid universal transverse mercator (UTM) zone |
| PGSGCT_E_BAD_DIRECTION | Invalid direction |
| PGSGCT_E_INVD_PROJECTION | Projection doesn't exists |
| PGSGCT_E_NO_POINTS | Number of points less than one |
| PGSGCT_E_GCTP_ERROR | Error in the GCTP library |
| PGSGCT_E_BAD_LONGITUDE | Bad longitude value (out of range) |
| PGSGCT_E_BAD_LATITUDE | Bad latitude value (out of range) |
| PGSGCT_W_INTP_REGION | Interrupted region encountered |

**EXAMPLES:**

**C:**

```
#include "PGS_GCT.h"

PGS_SMF_status    retValue;
PGSt_double       projParam[15] ;
PGSt_double       latitude[4];
PGSt_double       longitude[4];
PGSt_double       mapX[4], mapY[4];
```

```
PGSt_integer      ProjId = PGSd_UTM;
PGSt_integer      nPoints = 4;
PGSt_integer      directFlag, i;
PGSt_integer      zone[4] = {0, 0, 0, 0};

PGSt_integer      cucFileId;

/* All parameters must be initialized to zero */

for (i = 0; i<15) i++)
{
      projParam[i] = 0;
}

/* C array starts from 0 */

for (i = 1; i<5) i++)
{
      longitude[i-1] = PI/i;
      latitude[i-1] = PI/4;
}
cucFileId = 10999;
retValue =  PGS_CUC_cons(cucFileId,"CLRK80_MAJOR_AXIS",
&ProjParam[0]);
retValue = PGS_CUC_cons(cucFileId,"CLRK80_MINOR_AXIS",
&ProjParam[1]);
      ProjParam[5] = PI/2;
      ProjParam[6] = 3000000 /*(false easting in meters) */
      ProjParam[7] = 75000000 /* (false northing in meters)
*/

      directFlag = PGSd_GCT_FORWARD;
retValue =     PGS_GCT_Init (projId, projParam, directFlag);
retValue =     PGS_GCT_Proj(projId, directflag, nPoints,
                 latitude, longitude, mapX, mapY, zone);

      directflag = PGSd_GCT_INVERSE; (cartesian to
      geographical)
retValue =     PGS_GCT_Init (projId, projParam, directFlag);
retValue =     PGS_GCT_Proj(projId, directflag, nPoints,
                     latitude, longitude, mapX, mapY, zone);
```

**FORTRAN:**

```
implicit none

include     "PGS_GCT.f"
include     "PGS_GCT_12"
include     "PGS_SMF.f"
```

```
      integer                 PGS_GCT_Proj
      integer                 retValue
      double precision        projParam(15)
      double precision        latitude(4)
      double precision        longitude(4)
      double precision        mapX(4), mapY(4)
      integer                 ProjId
      integer                 nPoints
      integer                 directFlag, i
      integer                 zone(4)

      integer                 cucFileId

      ProjId = PGSd_UTM

      nPoints = 4

C     Projection parameters must be initialized to zero

      do 10 i= 1, 15

            projParam(i) = 0

10    continue

C     FORTRAN array starts from 1

      do 20 i= 1, 4

            longitude(i) = PI/i
            latitude(i) = PI/4

20    continue
      cucFileId = 10999
      pgs_cuc_cons(cucFileId,"CLRK80_MAJOR_AXIS", ProjParam(0));
      pgs_cuc_cons(cucFileId,"CLRK80_MINOR_AXIS", ProjParam(1));
      ProjParam(5) = PI/2

C     false easting in meters
      ProjParam(6) = 3000000

C     false northing in meters
      ProjParam(7) = 75000000

      directFlag = PGSd_GCT_FORWARD
      retValue =     PGS_GCT_Init (projId, projParam, directFlag)
      retValue =     PGS_GCT_Proj(projId, directflag, nPoint
                 latitude, longitude, mapX, mapY, zone)

C     cartesian to geographical
```

```
directflag = PGSd_GCT_INVERSE
pgs_gct_init (projId, projParam, directFlag)
pgs_gct_proj(projId, directflag, nPoints,
             latitude, longitude, mapX, mapY, zone)
```

**NOTES**: The units of output cartesian coordinates essentially depends on the units used for the Earth's radii, false easting and northing, etc., in the parameters list. The only requirement is that the units used should be consistent.

The zones[] parameter is at present only used for UTM transformations. It's an output parameter in the FORWARD direction and input parameter in the INVERSE direction.

All points are processed even if there is an error condition for some points. If bad point(s) are encountered the routine returns PGSd_GCT_IN_ERROR in the output vector. The user can find out the offending input values by searching for the PGSd_GCT_IN_ERROR in the output vector. For example, if the third point is in error then:

| | |
|---|---|
| Input Vector | |
| Longitude | 1, 2, 3, 4, 5 |
| latitude | 1, 1, 1, 1, 1 |
| Output Vector | |
| X | .01, .02, PGSd_GCT_IN_ERROR, .04, .05 |
| Y | .1, .2. PGSd_GCT_IN_ERROR, .4, .5 |

For the inverse transformations, two projections, namely Interrupted Goode and Interrupted Mollweide sometimes encounter a point that is in an interrupted region. In such cases the tool does not abandon processing but puts a value PGSd_GCT_IN_BREAK in the output vector. At the end of processing the tool returns a warning that an Interrupted region was encountered. The user can find out the offending input values by searching for the PGSd_GCT_IN_BREAK in the output vector. For example, if the third point is in the interrupted region:

| | |
|---|---|
| Input Vector | |
| X | 1, 2, 3, 4, 5 |
| Y | 1, 1, 1, 1, 1 |
| Output Vector | |
| Longitude | .01, .02, PGSd_GCT_IN_BREAK, .04, .05 |
| latitude | .1, .2. PGSd_GCT_IN_BREAK, .4, .5 |

**REQUIREMENTS**: PGSTK–1500, 1502

### 6.3.6 Math and Statistical Support Tools

IMSL has been selected to provide a suite of standard mathematical manipulation functions in a uniform package. This package is available to SCF and DAAC facilities through the EDHS server. Usage of the functionality supplied by the math package will not be mandatory and user developed or shareware routines may be included in science processing software. Users will be responsible for the functionality and long term maintenance of their homegrown software. IMSL service will be provided gratis.

We note that internal Toolkit software does not depend on IMSL.

### 6.3.7 Constants and Unit Conversions

#### 6.3.7.1 Introduction

The constants and unit conversion tools provide a means to access commonly used mathematical and physical constants, and a coherent means to perform unit conversions and parameter translations.

When the units conversion required is a linear conversion (e.g., degrees to radians) the most efficient mechanism for the programmer is to be given access to a physical constant that describes that transformation and then for the programmer to use this as appropriate. Providing a calling routine to perform the transformation would be inappropriate to most unit conversions and therefore specific API's for this are not provided.

#### 6.3.7.2 Requirements Compliance

a. PGSTK–1521 states that the Toolkit shall provide a means of accessing constant values related to an instrument. Constants that relate to instrument parameters are treated as ancillary data (static internal and dynamic internal) to the algorithms. The mechanism for retrieving instrument ancillary data is described elsewhere in section 6.2.1.6; this requirement is fulfilled by that tool.

b. An external file using some standard parameter=value mechanism will be used to store the mathematical and physical constants, in a similar way as performed in the ancillary data access tools. In this way the values are capable of adjustment without recompilation (requirement PGSTK–1522)

c. PGSTK–1531 states that unit conversion tools shall transform multiple values in a single call. As described above, the most efficient way for the programmer to perform unit conversions is to be given access to the conversion factor, which will be provided by the following tool. This requirement is therefore redundant.

## Obtain a Value for a User Specified Constant

**NAME:** **PGS_CUC_Cons**

**SYNOPSIS:**

C:

```
#include <PGS_CUC.h>

PGSt_SMF_Status
PGS_CUC_Cons (
        PGSt_integer        inpfileid,
        char                *inpParameter,
        PGSt_double         *outvalue)
```

FORTRAN:

```
include'PGS_CUC_11.f'
include'PGS_SMF_.f'

integer function PGS_CUC_Cons (inpfileid, inpParameter, outvalue)
        integer             inpfileid,
        character           inpParameter,
        double precision    outvalue)
```

**DESCRIPTION:** This routine receives the fileid and the constant name from the user. The fileid allows more than one input file to be used, thus allowing the user to implement his or her own specialized input files with constants. The parameter is a character string representing the constant whose numerical value is sought by the user. The resulting value is passed back to the user.

**INPUTS:**

***Table 6-256. PGS_CUC_Cons Input***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| fileid | file identifier | N/A | N/A | N/A |
| parameter | constant wanted | N/A | N/A | N/A |

**OUTPUTS:**

***Table 6-257. PGS_CUC_Cons Output***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| value | constant value | N/A | N/A | N/A |

**RETURNS:**

*Table 6-258. PGS_CUC_Cons Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGSCUC_E__ERROR | error in finding the constant value |
| The following are returned to the error log: | |
| PGSCUC_E_CANT_GET_FILE_ID | |
| PGSCUC_E_CANT_OPEN_INPUT_FILE | |
| PGSCUC_E_AGG_CANT_BE_INSERTED | |
| PGSCUC_E_READLABEL_PARSE_ERROR | |
| PGSCUC_E_PARAMETER_INVALID | |
| PGSCUC_E_FIRST_NODE_NOT_FOUND | |

**EXAMPLES:**

C:

```
char parameter[] = {"PI"};
int  fileid = 19701;
double result;

ret_status = PGS_CUC_Cons(parameter, fileid, result);
```

FORTRAN:

```
implicit none

include     'PGS_CUC_11.f'
include     'PGS_SMF.f'

integer           pgs_CUC_cons
integer           ret_status
integer           inpfileid
character*100     inpParameter
double precision  outvalue
inpfileid = 10790
inpParameter = 'pi'

ret_status = pgs_cuc_cons(inpfileid, inpParameter, outvalue)
ret_value = PGS_CUC_get_parameter("pi", pi)
```

**NOTES:** User defines key word to be searched for within a logical file. User also defines fileid so that location of file can be found. Tool uses ODL libraries to conduct a parameter equals value search. For further information see Constant and Unit Conversions (CUC) Tools Primer, Object Description Language (ODL) documentation.

**REQUIREMENTS:** PGSTK–1520, PGSTK–1521, PGSTK–1522, PGSTK–1530

## Obtain Slope and Intercept to Calculate Conversion Between Specified Units.

**NAME:** **PGS_CUC_Conv**

**SYNOPSIS:**

C:

```
#include <PGS_CUC.h>

PGSt_SMF_Status
PGS_CUC_Conv (
        char            inpUnit[],
        char            outUnit[],
        PGSt_double     *outSlope,
        PGSt_double     *outIntercept)
```

FORTRAN:

```
include'PGS_CUC_11.f'
include'PGS_SMF.f'

integer function PGS_CUC_Conv(inpUnit, outUnit, outSlope,
                                        outIntercept)
        character*100       inpunit,
        character*100       outunit,
        pgst_double         outslope,
        pgst_double         outintercept)
```

**DESCRIPTION:** This routine receives two character descriptions of Units as inputs. The first input is the unit that the user has; the second input being the unit the user wants to transform to. Both Unit descriptions are held in a file, after a search identifies whether each unit is held within the file the slope and intercept of the conversion between units is calculated. The resulting values for slope and intercept are then passed back to the user.

**INPUTS:**

***Table 6-259. PGS_CUC_Conv Inputs***

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| inpUnit | unit you have | N/A | N/A | N/A |
| outUnit | unit you want | N/A | N/A | N/A |

**OUTPUTS:**

*Table 6-260. PGS_CUC_Conv Outputs*

| Name | Description | Units | Min | Max |
|---|---|---|---|---|
| outSlope | mathematical slope | N/A | N/A | N/A |
| outIntercept | mathematical intercept | N/A | N/A | N/A |

**RETURNS:**

*Table 6-261. PGS_CUC_Conv Returns*

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Successful return |
| PGS_E_CUC_ERROR | error in performing conversion match |
| The following are returned to the error log: | |
| PGSCUC_E_COULDNT_INIT_UDUNITS3 | |
| PGSCUC_E_DONT_KNOW_INP_UNIT | |
| PGSCUC_E_DONT_KNOW_OUTP_UNIT | |
| PGSCUC_E_UNITS_ARE_INCOMPATIBLE | |
| PGSCUC_E_A_UNIT_IS_CORRUPTED | |

**EXAMPLES:**

C:

```
char inpUnit[] = {"centigrade"};
char outUnit[] = {"fahrenheit"};
double *outSlope;
double *outIntercept;

ret_status = PGS_CUC_Conv(inpUnit, outUnit, outSlope,
outIntercept);
```

FORTRAN:

```
implicit none

include     'PGS_CUC_11.f'
include     'PGS_SMF.f'

integer           PGS_CUC_Conv
integer           ret_status
character*100     inpUnit
character*100     outUnit
double precision  outSlope
double precision  outIntercept
InpUnit = 'metres'
OutUnit = 'feet'
ret_status = pgs_cuc_conv(inpUnit, outUnit, outSlope,
>                             outIntercept)
```

**NOTES:** For further details on this tool, see Appendix I. Background on library units used, Units available for conversion and adding own conversion Units to the file.

**REQUIREMENTS:** PGSTK–1520, PGSTK–1521, PGSTK–1522, PGSTK–1530

(C) Copyright 1992 UCAR/Unidata

Permission to use, copy, modify, and distribute this software and its documentation for any purpose without fee is hereby granted, provided that the above copyright notice appears in all copies, that both that copyright notice and this permission notice appear in supporting documentation, and that the name of UCAR/Unidata not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. UCAR makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. It is provided with no support and without obligation on the part of UCAR or Unidata, to assist in its use, correction, modification, or enhancement.

### 6.3.8 Dynamic Memory Management Tools

## Allocate Memory

**NAME:** **PGS_MEM_Malloc( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Malloc(
        void     **addr,
        size_t   numBytes);
```

FORTRAN: None

**DESCRIPTION:** This tool allocates an arbitrary number of bytes in memory.

**INPUTS:** numBytes—number of bytes to allocate

**OUTPUTS:** addr—pointer to beginning of address that has been allocated

**RETURNS:**

***Table 6-262. PGS_MEM_Malloc Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSMEM_E_NO_MEMORY | No memory space available for current process |
| PGSMEM_W_MEMORY_USED | Memory address has been allocated previously |

**EXAMPLES:**

```
int                 i;
int                 *intPtr = (int *)NULL;
PGSt_SMF_status     returnStatus;

returnStatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10);
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

**NOTES:** This tool will control the amount of memory that may be allocated at any one time. You should call PGS_MEM_Free( ) to free the memory allocated once you are done using it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process

**REQUIREMENTS:** PGSTK–1240

# Allocate Memory

**NAME:** **PGS_MEM_Calloc( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Calloc(
        void     **addr,
        size_t   num_elems,
        size_t   elem_size);
```

FORTRAN: None

**DESCRIPTION:** This tool allocates an arbitrary number of bytes in memory. All bytes of the allocated memory will be initialized to zero.

**INPUTS:** num_elems—number of elements

elem_size—size of the element in bytes

**OUTPUTS:** addr—pointer to beginning address of the memory that has been allocated

**RETURNS:**

***Table 6-263. PGS_MEM_Calloc Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSMEM_E_NO_MEMORY | No memory space available for current process |
| PGSMEM_W_MEMORY_USED | Memory address has been allocated previously |

**EXAMPLES:**

```
int                 i;
int                 *intPtr = (int *)NULL;
PGSt_SMF_status     returnStatus;

returnStatus = PGS_MEM_Calloc((void
      **)&intPtr,10,sizeof(int));
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

**NOTES:** This tool will control the amount of memory that may be allocated at any one time. You should call PGS_MEM_Free( ) to free the memory allocated once you are done with it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, that have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

**REQUIREMENTS:** PGSTK–1240, PGSTK–1241

# Re–Allocate Memory

**NAME:** **PGS_MEM_Realloc( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

PGSt_SMF_status
PGS_MEM_Realloc(
        void     **addr,
        size_t   newsize);
```

FORTRAN: None

**DESCRIPTION:** This tool reallocates the number of bytes requested.

**INPUTS:** addr—pointer to the starting address of previously allocated memory

newsize—new total memory size to reallocate

**OUTPUTS:** addr—pointer to starting address of newly allocated memory

**RETURNS:**

***Table 6-264. PGS_MEM_Realloc Returns***

| Return | Description |
|---|---|
| PGS_S_SUCCESS | Success |
| PGSMEM_E_NO_MEMORY | No memory space available for current process |
| PGSMEM_E_ADDR_NOTALLOC | Address is not allocated previously |

**EXAMPLES:**

```
int                 i;
int                 *intPtr = (int *)NULL;
PGSt_SMF_status     returnStatus;

returnStatus = PGS_MEM_Calloc((void
      **)&intPtr,10,sizeof(int));
if (returnStatus == PGS_S_SUCCESS)
{
   for (i=0 ; i < 10 ; i++)
   {
      intPtr[i] = i;
   }
}
```

```
returnStatus = PGS_MEM_Realloc((void
      **)&intPtr,sizeof(int)*20);
if (returnStatus == PGS_S_SUCCESS)
{
   # Realloc success #
}
```

**NOTES:** This tool will control the amount of memory that needs to be reallocated to a pointer that has already been used to obtain an initial allocation of memory through one of the available Toolkit routines. You should call PGS_MEM_Free( ) to deallocate the memory once you are done using it; failure to do so may cause future memory allocation requests to fail within the same process.

Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, that have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

**REQUIREMENTS:** PGSTK–1240

# Initialize Memory to Zero

**NAME:** **PGS_MEM_Zero( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_Zero(
        void    *addr,
        size_t  numBytes);
```

FORTRAN: None

**DESCRIPTION:** This tool initializes a memory block or structure to zero.

**INPUTS:** addr—beginning address of the memory block or structure

numbytes—number of bytes

**OUTPUTS:** None

**RETURNS:** None

**EXAMPLES:**

```
Typedef     struct
{
    int     i;
    char    c;
    float   f;
}TestStruct;

TestStruct        test
int               *intptr = (int *)NULL
returnstatus      returnstatus

PGS_MEM_Zero(&test,sizeof(test));
returnstatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10 );
if (returnstatus == PGS_S_SUCCESS)
{
    PGS_MEM_Zero(intPtr,sizeof(intPtr)*10)
}

PGS_MEM_Zero( s, sizeof(longint)*10 );
```

**REQUIREMENTS:** PGSTK–1240

# De–Allocate Memory

**NAME:** **PGS_MEM_Free( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_Free(
        void    *addr);
```

FORTRAN: None

**DESCRIPTION:** This tool deallocates memory that was previously allocated through the use of a Toolkit allocation routine.

**INPUTS:** addr—address of previously allocated memory

**OUTPUTS:** None

**RETURNS:** None

**EXAMPLES:**

```
int                 *intPtr = (int *)NULL;
returnStatus        returnStatus;

returnStatus = PGS_MEM_Malloc((void
      **)&intPtr,sizeof(int)*10);
if (returnStatus == PGS_S_SUCCESS)
{
    PGS_MEM_Free(intPtr);
    intPtr = (int *)NULL;
}
```

**NOTE**: Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to re–use. As a reminder, ALL local pointer variables MUST be initialized to NULL prior to use. Failure to heed these warnings may result in anomalous behavior within your process.

**REQUIREMENTS:** PGSTK–1240

# De–Allocate Memory

**NAME:** **PGS_MEM_FreeAll( )**

**SYNOPSIS:**

C:

```
#include <PGS_MEM.h>

void
PGS_MEM_FreeAll(
        void);
```

FORTRAN: None

**DESCRIPTION:** Deallocates all memory that was previously allocated through the use of Toolkit allocation routines, within an executable. Calls to PGS_MEM_Free( ) and PGS_MEM_FreeAll( ) may be interlaced.

**INPUTS:** None

**OUTPUTS:** None

**RETURNS:** None

**EXAMPLES:**

```
typedef struct
{
   int      i;
   char     c;
   float    f;
}TestStruct;

TestStruct  *test       = (TestStruct *)NULL;
int         *intPtr     = (int *)NULL;

PGS_MEM_Malloc((void **)&intPtr,sizeof(int)*10);
PGS_MEM_Malloc((void **)&test,sizeof(TestStruct)*10);
PGS_MEM_FreeAll();

test        = (TestStruct *)NULL
intPtr      = (int *)NULL
```

**NOTES**: This tool should only be called near the end of processing, or when no further allocation of dynamic memory will be required.

Due to the comprehensive nature of this tool, all allocated memory references, that have not yet been freed, will be disposed of. Because the Toolkit memory functions track memory usage, it is imperative that pointer variables, which have been freed, be initialized to NULL prior to

use. Failure to heed these warnings may result in anomalous behavior within your process.

**REQUIREMENTS:** PGSTK–1240

### 6.3.9 Graphics Support Tools

These tools will support the analysis of graphics, quicklook and quality assurance (QA) data output from science production processes. It is assumed that the exchange format of the data files will be HDF, although specific graphics formats are TBD at the time of this document. These tools will contain an image processing capability, which will be used in conjunction with the math library chosen for the Toolkit (section 6.3.6), or with user supplied math functions. It is expected that this functionality will be a subset of the data visualization capability supplied by EOSVIEW, a package being developed to display EOS data structure.