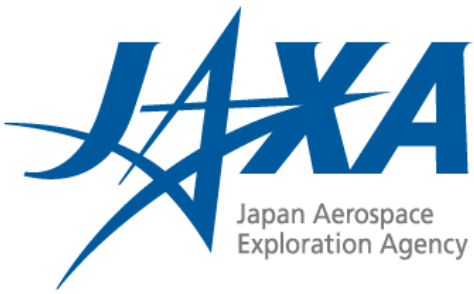
Data Users' Manual for the Advanced Microwave Scanning Radiometer 2 (AMSR2) onboard the Global Change Observation Mission 1st - Water "SHIZUKU" (GCOM-W1)

4th Edition November 2016

Japan Aerospace Exploration Agency
Earth Observation Research Center



Change record

Issue	Date (YYYY/MM/DD)	Description of change
1st Edition	2013/03/19	First Release
2nd Edition	2013/03/22	“Read L2/L3 Product” is added.
3rd Edition	2016/03/25	SST changed from 1 layer to 2 layers.
4th Edition	2016/11/17	Sample program to read “L1B/L1R Pixel Data Quality” is changed.

Contents

1. How to Get Related Manuals and Products

This document provides users with information about the method for utilization of GCOM-W1/AMSR2 data sets as well as the background knowledge. For more details, please visit the “GCOM-W1 Data Providing Service” and download the related documents which are shown below.

GCOM-W1 Data Providing Service
`http://gcom-w1.jaxa.jp/`

- GCOM-W1 "Shizuku" Data Users Handbook
- AMSR2 Level 1 Product Format Specification
- AMSR2 Higher Level Product Format Specification (L3)

“GCOM-W1 Data Providing Service” provides products derived from data obtained by the AMSR2 instrument onboard the GCOM-W1 (SHIZUKU), AMSR onboard the ADEOS-II (Midori II) and the AMSR-E onboard the Aqua satellite with free of charge for research and educational purposes. User registration is required to use the products.

AMSR2 Product I/O Toolkit (AMTK), User's Manual, and leap seconds data are also available from “GCOM-W1 Data Providing Service”. For more information related to the GCOM-W1 satellite, please visit the GCOM website.

GCOM website
`http://suzaku.eorc.jaxa.jp/GCOM/`

In this document, we also explain about the Hierarchical Data Format version 5 (HDF5). Please download HDF5 libraries and SZIP library from The HDF Group website.

The HDF Group
`http://www.hdfgroup.org/`

Sample programs which we explain in this document are also available from the website below.

`http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_use.html`

2. Overview of this Data Users Manual

To make effective use of the GCOM-W1/AMSR2 data sets, firstly we will explain some basic information of the GCOM-W1 satellite and AMSR2 instrument. Then we will explain about how to install the libraries and use sample programs. Sample programs both written in C and Fortran Languages are available.

The AMSR2 products store data as hierarchical data format (HDF). The HDF file format has two versions; HDF4 and HDF5. The HDF5 file format is used in AMSR2 products. If you want to read the data in HDF5 format by C or Fortran programs, you should install the HDF5 library on your computer. There are two ways to read AMSR2 products; 1) to use the HDF5 library only; and 2) to use the AMSR2 Product I/O Tool Kit (AMTK) which uses the HDF5 library as internal routine.

This document explains how to access the AMSR2 products in those two ways -- with or without using AMTK. Also, the same contents are explained in both C and Fortran programs, so please select the best method according to your execution environment from below:

- C Programming with AMTK
- FORTRAN90 Programming with AMTK
- C Programming with HDF5 library (without AMTK)
- FORTRAN90 Programming with HDF5 library (without AMTK)

AMTK provides several functions for reading and storing AMSR2 products.

Please note that scanning time data is stored as TAI93 in the HDF5 file. AMTK will internally convert TAI93 to year, month, day, hour, minute, second, and millisecond, when users read data.

Latitude and longitude information at observation points in a scan for only 89GHz channel are stored in HDF5 file. AMTK reads 89GHz channel latitude and longitude, and calculates latitude and longitude of each lower frequency channels from them.

Some data is converted with a scale factor when it is stored into the HDF5 file. AMTK converts stored value to original value automatically using a scale factor that stored inside the HDF5 file.

As sample program, we also prepared convert routines to read scan time and latitude/longitude information for lower frequency channels to whom use only the HDF5 library in their program codes.

3. Basic Knowledge of GCOM-W1/AMSR2 Products

3.1 Satellite Orbit

Orbit of the GCOM-W1 satellite is same as that of the Aqua satellite, which carries the AMSR-E instrument, called “A-train” orbit. Table 1 shows the major characteristics of the GCOM-W1 satellite, and Figure 1 provides the schematic image of the GCOM-W1 orbit.

Table 1. Major Characteristics of the GCOM-W1 Satellite

Item	Characteristics
Orbit	Sun-synchronous sub-recurrent
Altitude (above the equator)	699.6km
Inclination	98.186°
Local sun time at ascending node	p.m. 1:30±15 min.
Revolutions per Recurrent Period	233 rev./16 days

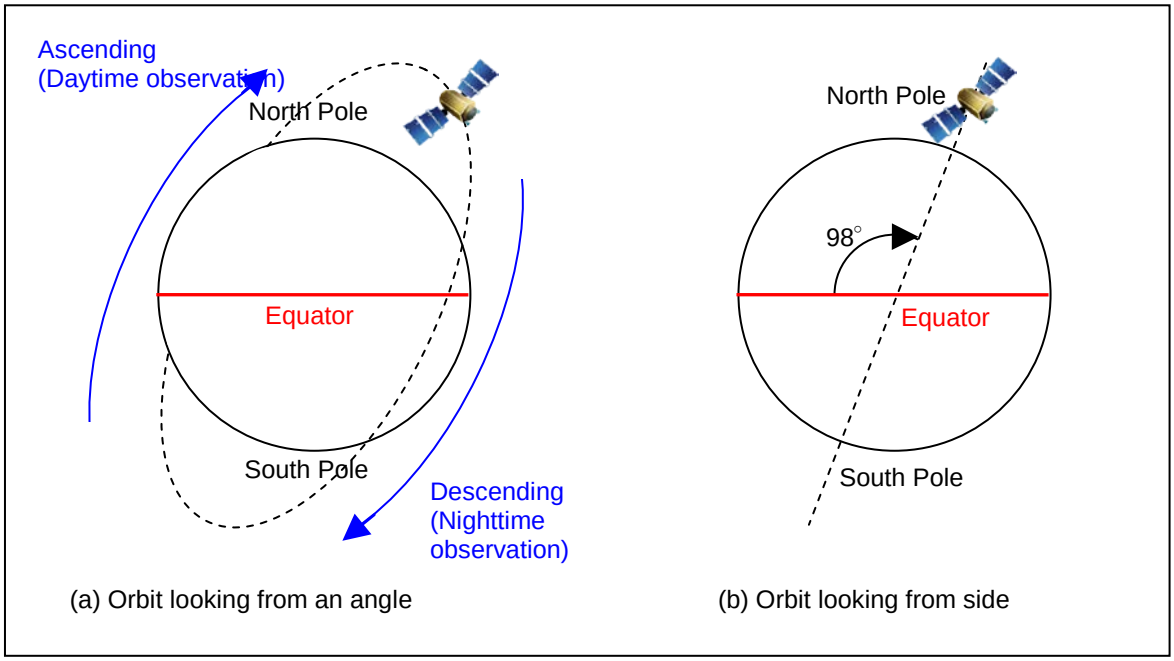


Figure 1. GCOM-W1 Satellite Orbit

GCOM-W1 is in a circular orbit of satellite inclination angle of 98 degree.

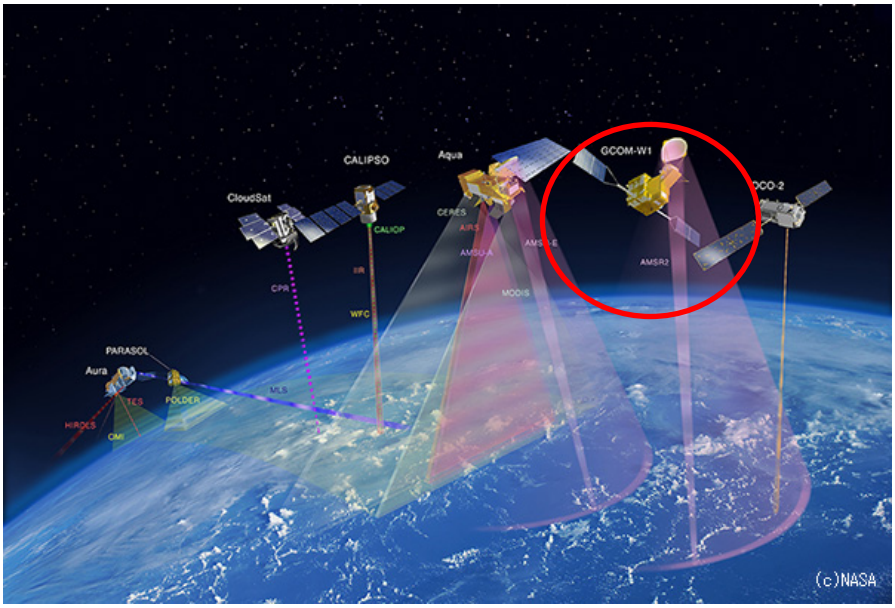
Orbit from the South Pole to the North Pole is called *ascending* path, and that from the North Pole to the South Pole is called *descending* path.

The GCOM-W1 satellite orbit is a sun-synchronous sub-recurrent, which is a combination of sun-synchronous orbit and sub-recurrent orbit.

In a sun-synchronous orbit, positioning between the satellite and the sun is always the same. Ascending path observes the day scene and descending path observes the night scene. Therefore, region where the satellite flies over always gets same sunlight angle. Ascending path is crossing the equator at around 1:30 p.m. local mean solar time.

In a sub-recurrent orbit, the satellite repeats its original orbit after a certain number of days. In the case of GCOM-W1 satellite, it observes the same area at 16 days interval, while flying around the earth 233 times, which is about 15 times a day.

A-Train



What is A-Train?

The Afternoon Constellation (A-Train) is a NASA-directed constellation of Earth observation satellites in a sun-synchronous orbit at an altitude of approximately 700 km that crosses the equator each day at about 1:30 p.m. Currently, with the A-train, the following satellites are participating: Aqua (NASA, U.S.), CloudSat (NASA), CALIPSO (NASA and CNES, France) and Aura (NASA) and Japan has participated in the system for the first time with GCOM-W1.

Special Features of the A-Train

For Earth observation, it is very efficient to perform observations by measuring the same one location with various sensors at the same time. With various satellites lining up on the almost same orbit, the A-Train enables us to observe the same location on the Earth by multiple satellites around the same time (within approximately 10 minutes.) The position of each satellite is strictly controlled; therefore, a new comer has to be injected into a pre-determined location that does not interfere with other already-flying members. GCOM-W1 entered the A-Train orbit successfully by utilizing JAXA's rendezvous technology. It is the first experience for JAXA to operate a satellite in the constellation flying on the almost same orbit.

A-Train (Satellite constellation) member satellites

Aura (NASA) (launched on July 15, 2004)

To acquire observational data for elucidating the composition of the earth atmosphere, its chemical react, and dynamics.

CALIPSO (NASA/CNES) (launched on April 28 2006)

An optical lidar satellite to acquire observational data to clarify impact of aerosol and clouds on the Earth's climate

CloudSat (NASA) (launched on April 28, 2006)

A radio wave radar satellite to acquire observational data to study the impact of clouds on the Earth's climate

Aqua (NASA) (launched on May 4, 2002)

The name came from the Latin word "Aqua" meaning water. The satellite acquires observational data on the Earth's various water circulations including water vapor in the atmosphere and from the ocean, clouds, precipitation, ocean ice, and ground water.

OCO-2 (NASA) (will be launched in 2014)

Consist of three high-resolution grating spectrometers for global measurements of atmospheric carbon dioxide (CO2).

3.2 Granules

To produce the GCOM-W1/AMSR2 standard products, observational data is sectioned into pieces, and each piece is called a "granule". Each granule is defined as a half orbit between the North Pole and the South Pole. Ascending scene is half orbit scans from the southern most point to the northern most point, and descending scene is orbit scans from the northern most point to the southern most point. Ascending scene contains the daytime observation, and descending scene contains the nighttime observation.

Figure 2 provides the sample image of observation area of AMSR2 standard products. Observed areas are shown in color and the white indicates unobserved area.

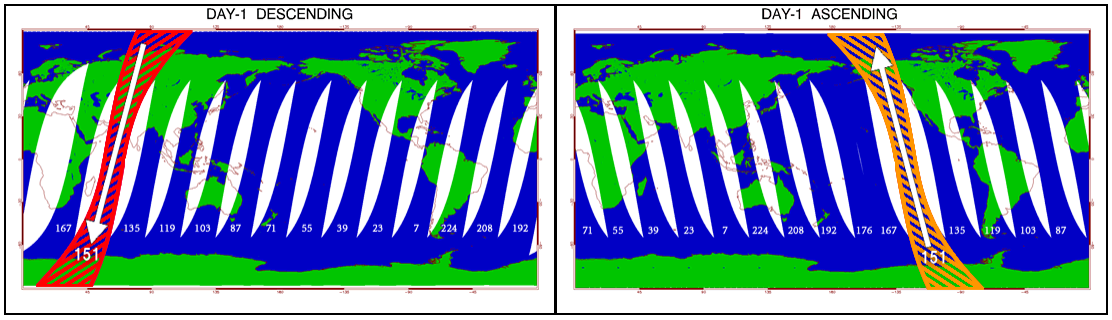


Figure 2. Granules of standard products

Numbers displayed in observed area are path numbers. Path numbers are determined for all the satellite ground tracks during a recurrent period (16 days) in a westward direction. Total number of paths are 233. Path number of contiguous paths increases by 16. For instance, product with path number of 151 distinguished from ascending path to descending path, since each granules of standard products are defined as a half orbit. The former is called “151A” and the latter is called “151D”. Area which is indicated in red at Figure 2 is the observation swath of the “151D” granule and orange area is the observation swath of the “151A” granule. Definition of granules and path numbers are as same as Aqua/AMSR-E.

Inconsistency between the definitions path number and granule

One path has a cycle that begins at an equator crossing point in the ascending node, but a data granule of AMSR2 is defined as a half of a revolution ranging from pole to pole. AMSR2 product file name use the path number where the data granule begins.

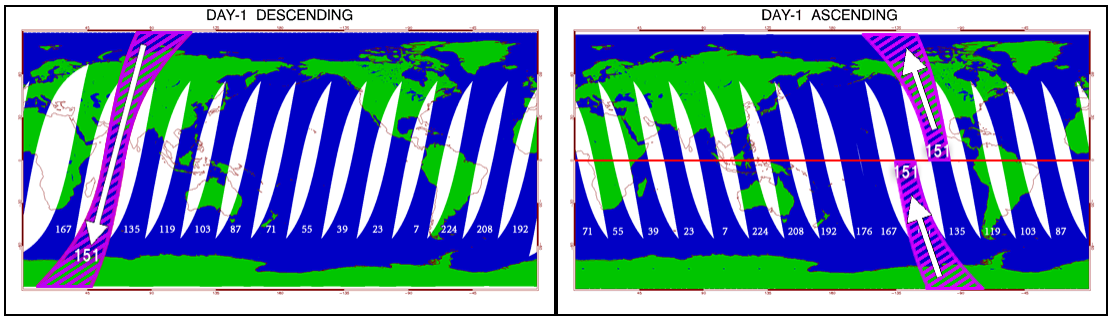


Figure 3. Difference between path number 151 and granule of 151

For example, Figure 3 provides the image of observation area of path number 151, which is colored in purple.

Path number is 151 *at northern most point* of granule "**151D**".

Path number is 151 *at southern most point* of granule "**151A**".

3.3 Level1 and Higher Level Products

AMSR2 is a multi-frequency microwave radiometer. It measures weak microwave emission (6.9 - 89.0GHz) from the surface and the atmosphere of the Earth. AMSR2 products are categorized according to the processing levels. Level 1A (L1A) products consist of raw observation counts, antenna temperature conversion coefficients, etc... Level 1B (L1B) products contain brightness temperatures which is converted from L1A count values using the conversion coefficients. Level 2 (L2) products contain geophysical parameters which primarily related to water and derived from the L1B products. AMSR2 derived geophysical parameters are shown in Table 2. L1 and L2 is swath data with geolocation information.

Table 2. AMSR2 retrieved geophysical parameters

Geophysical Parameters Name
Total Precipitable Water(TPW)
Cloud Liquid Water(CLW)
Precipitation(PRC)
Sea Surface Temperature(SST)
Sea Surface Wind Speed(SSW)
Sea Ice Concentration(SIC)
Snow Depth(SND)
Soil Moisture Content(SMC)

Level 3 (L3) products are global gridded products at two different resolutions (0.1 and 0.25 degrees). Products make ground projections on the globe and in the North Pole and South Pole regions by taking the time and spatial averages of the L1B and L2 standard products. L3 products of brightness temperatures are called L3TB and those of geophysical parameters are called L3GEO. Daily and monthly statistical products are created for both ascending and descending data sets. L2 and L3GEO products are also called as higher level products.

Figure 4 shows the relationship between swath data and global data.

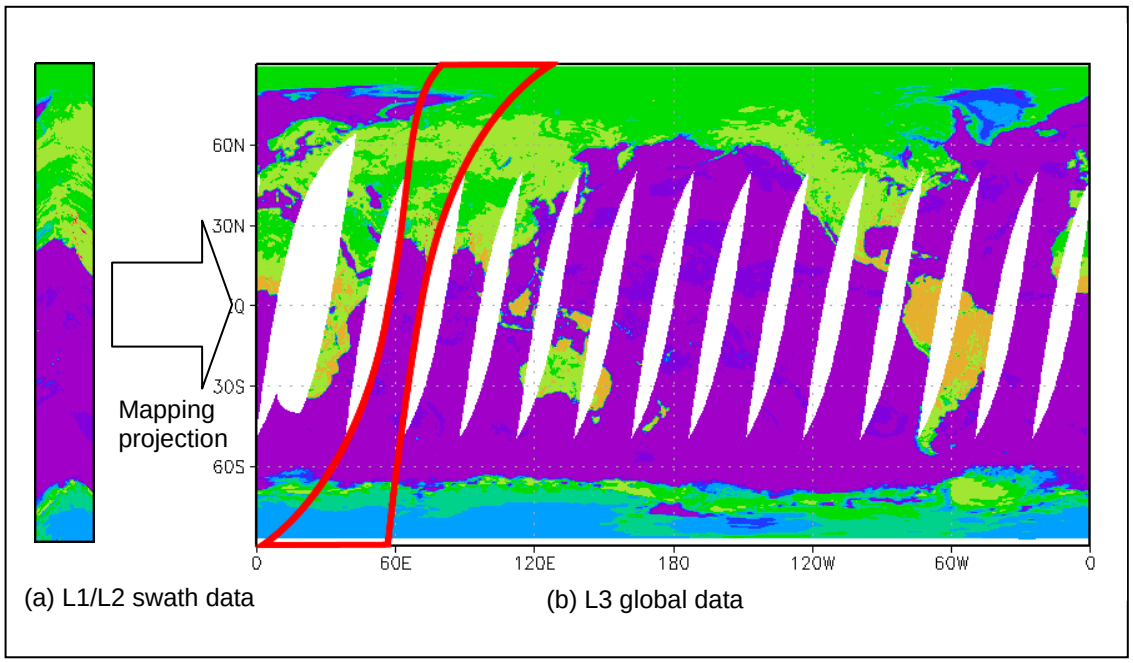


Figure 4. Relationship between swath and global data

3.4 File Name Convention

AMSR2 product file name is ruled as below.

File name = Granule ID + extension [.h5]

Naming rules of L1/L2 and L3 granule ID are shown in Figure 5 and Figure 6, respectively.

```
                                1           2           3           4
Byte Location:   12345678901234567890123456789012345678901
AMSR2            :   GW1AM2_YYYYMMDDHHmm_PPPX_LLxxKKKrdvaaappp
L1 Sample        :   GW1AM2_201209071216_068D_L1SGBTBR_0000000
L2 Sample        :   GW1AM2_201209071216_068D_L2SGSSTLA0000000

GW1 : Satellite(Fixed Value)   AM2 : Sensor(Fixed Value)   YYYYMMDDHHmm : Observation Start Time (UTC)
PPP : Path Number(001～233)   X : Orbit direction(A: Ascending, D: Descending, B: Both)
LL   : Process Level(L1:Level 1, L2:Level 2)   xx : Process Kind(SG: Standard operation product, SN: Near real time operation product (Global), SL: Near real time operation product (Local))
KKK : Product ID
        L1 : ADN -> Digital Number(L1A),          BTB -> Brightness Temperature(L1B),
             RTB -> Brightness Temperature (L1R)
        L2 : CLW -> Cloud Liquid Water                PRC -> Precipitation
             SIC -> Sea Ice Concentration             SMC -> Soil Moisture Content
             SND -> Snow Depth                            SST -> Sea Surface Temperature
             SSW -> Sea Surface Wind Speed            TPW -> Total Precipitable Water
r    : Resolution
        L1 : R -> Raw(Fixed),
        L2 : L -> Low(Small number of observation point(243 points))
             H -> High(Large number of observation point(486 points))
d    : Developer ID
        L1 : _ -> Underscore(Fixed)
        L2 : A-Z
v    : Version number of Product (0-9, A-Z)
aaa : Version number of algorithm (000-999)   ppp : Version number of processing parameter (000-999)
```

Figure 5. L1 and L2 granule ID

```
                                1         2         3         4
Byte Location     :  12345678901234567890123456789012345678901
AMSR2             :   GW1AM2_YYYYMMDD_ttt_PPWX_LLxxKKKrdvaaappp
L3 Sample(GEO)    :  GW1AM2_20120907_01D_EQOA_L3SGSSTLA0000000
L3 Sample(TB)     :  GW1AM2_20120907_01D_EQMA_L3SGT06LA0000000

GW1 : Satellite(Fixed Value)   AM2 : Sensor(Fixed Value)
YYYYMMDD : Start date of observation (UTC), "DD"＝"00" means monthly data
ttt : Calculating period (01D:Daily, 01M:Monthly)
PP  : Map projection (EQ: Equirectangular, PN: Polar stereo(North hemisphere), PS: Polar stereo(South
hemisphere))
W   : Calculating method (M: Mean, O: Overwrite)   X : Orbit direction(A: Ascending, D: Descending)
LL  : Process Level (L3:Level 3)   xx : Process Kind(SG: Standard operation product)
KKK : Product ID
          T06, T07, T10, T18, T23, T36, T89 -> Brightness Temperature(H and V are stored in 1
product)
          CLW -> Cloud Liquid Water              PRC -> Precipitation
          SIC -> Sea Ice Concentration           SMC -> Soil Moisture Content
          SND -> Snow Depth                      SST -> Sea Surface Temperature
          SSW -> Sea Surface Wind Speed          TPW -> Total Precipitable Water
r   : Resolution (L: Low(25km or 0.25deg), H: High(10km or 0.1deg))
```

Projection	Low resolution interval		High resolution interval	
	Width number [Longitude direction]	Length number [Latitude direction]	Width number [Longitude direction]	Length number [Latitude direction]
EQR(TB, SIC)	1440	720	3600	1800
PN(TB, SIC)	304	448	760	1120
PS(TB, SIC)	316	332	790	830
PN(SND)	432	574	1080	1435

```
d   : Developer ID
      L3 : A-Z
v   : Version number of Product (0-9, A-Z)
aaa : Version number of algorithm (000-999)   ppp : Version number of processing parameter (000-999)
```

Figure 6. L3 granule ID

3.5 Scans and Overlaps

Figure 7 shows AMSR2 observation concept, illustrating how the AMSR2 instrument is operated once in orbit. The AMSR2 sensor unit rotates counterclockwise about the Z-axis while in flight, creating a conical scan pattern in which the antenna beam transcribes an arc of approximately 1,660km in diameter on the Earth's surface. There are a number of observation points in each scan. L1 and L2 products contain earth observational data as two dimensional array.

Figure 8 shows schematic image of overlap scans. Nevertheless L1 and L2 granule is defined as a half of a revolution ranging from pole to pole, L1 product includes previous and next granule scans around the pole and extended about 20 scans at both end. L2 product doesn't stores overlap scans, so there is 40 scans difference between L1 and L2 product of same path.

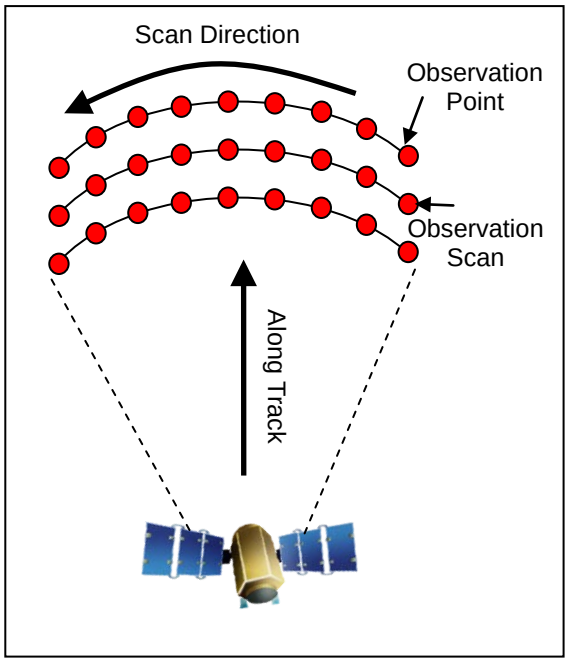


Figure 7. AMSR2 observation concept

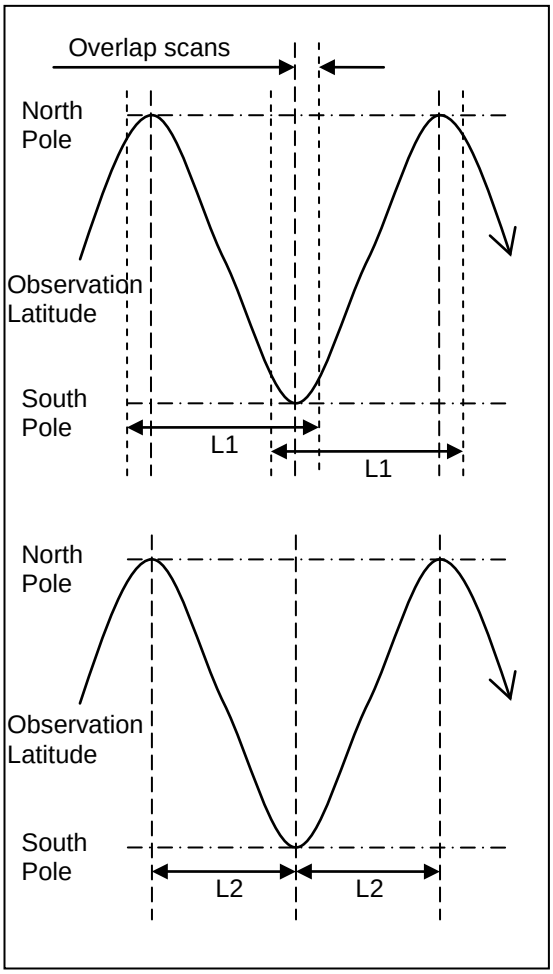


Figure 8. Image of overlap scans

3.6 Scan Time

Scanning time data is stored as TAI93 in AMSR2 L1 and L2 products. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.

A leap second is a second added to or subtracted from Coordinated Universal Time (UTC) to make it agree with astronomical time. Leap seconds are irregularly spaced because the Earth's rotation speed changes irregularly. To deal with TAI93 conversion, you need to know when a leap second is added to or subtracted from UTC.

For example, for the date of July 24, 2012, elapsed second time which doesn't include the leap second from January 1st, 1993 is 617,241,600. Leap seconds are added 8 times between these times, so TAI93 of this date is represented as 617,241,608.

With AMTK, scanning time data is automatically converted from TAI93 to year, month, day, hour, minute, second, and millisecond. If you only use the HDF5 library in your program codes, you need to convert scan time by yourself. (Subroutine is prepared in sample programs.)

3.7 Frequency and Observation Footprint

AMSR2 measures brightness temperatures at 6.9, 7.3, 10.7, 18.7, 23.8, 36.5, and 89.0 GHz frequency channels. Vertically and horizontally polarized measurements are taken at all channels.

There are two channels in 89 GHz (89 GHz-A and 89 GHz-B). A total of 16 channels are available. Table 3 shows the AMSR2 observation frequencies and polarizations.

Table 3. AMSR2 observation frequencies and polarizations

No.	Frequency/polarization	Notes
1	6.9GHz horizontal polarized channel	Low Frequency Channels: Number of observation points are 243.
2	6.9GHz vertical polarized channel	
3	7.3GHz horizontal polarized channel	
4	7.3GHz vertical polarized channel	
5	10.7GHz horizontal polarized channel	
6	10.7GHz vertical polarized channel	
7	18.7GHz horizontal polarized channel	
8	18.7GHz vertical polarized channel	
9	23.8GHz horizontal polarized channel	
10	23.8GHz vertical polarized channel	
11	36.5GHz horizontal polarized channel	
12	36.5GHz vertical polarized channel	
13	89.0GHz A horizontal polarized channel	High Frequency Channels: Number of observation points are 486.
14	89.0GHz A vertical polarized channel	
15	89.0GHz B horizontal polarized channel	
16	89.0GHz B vertical polarized channel	

6.9 - 36.5 GHz channels are low frequency channels. Number of observation points are 243 for each scan. 89.0GHz channels are high frequency channels, and number of observation points are 486 for each scan. Each geophysical parameter is derived from different channels, and number of observation points for each scan depends on channels which used mainly to retrieve parameter Table 4 shows the number of observation points for each L2 product scan.

Table 4. Number of observation points for each L2 product scan

Geophysical Parameter	No. of Observation Points
Total Precipitable Water(TPW)	243
Cloud Liquid Water(CLW)	243
Precipitation(PRC)	486
Sea Surface Temperature(SST)	243
Sea Surface Wind Speed(SSW)	243
Sea Ice Concentration(SIC)	243
Snow Depth(SND)	243
Soil Moisture Content(SMC)	243

The main beam’s shape and distance on the earth surface is called the “footprint”. It depends on satellite altitude, the off-nadir angle, and the beam width. The footprint’s shape is determined by the Earth’s curvature. Table 5 and Figure 9 show the footprint size and schematic image of each frequency channel.

Table 5. Footprint

Frequency	Footprint (along scan x along track)
6.9GHz	35km x 62km
7.3GHz	34km x 58km
10.7GHz	24km x 42km
18.7GHz	14km x 22km
23.8GHz	15km x 26km
36.5GHz	7km x 12km
89.0GHz	3km x 5km

* Footprint of 18.7GHz is smaller than that of 23.8GHz.

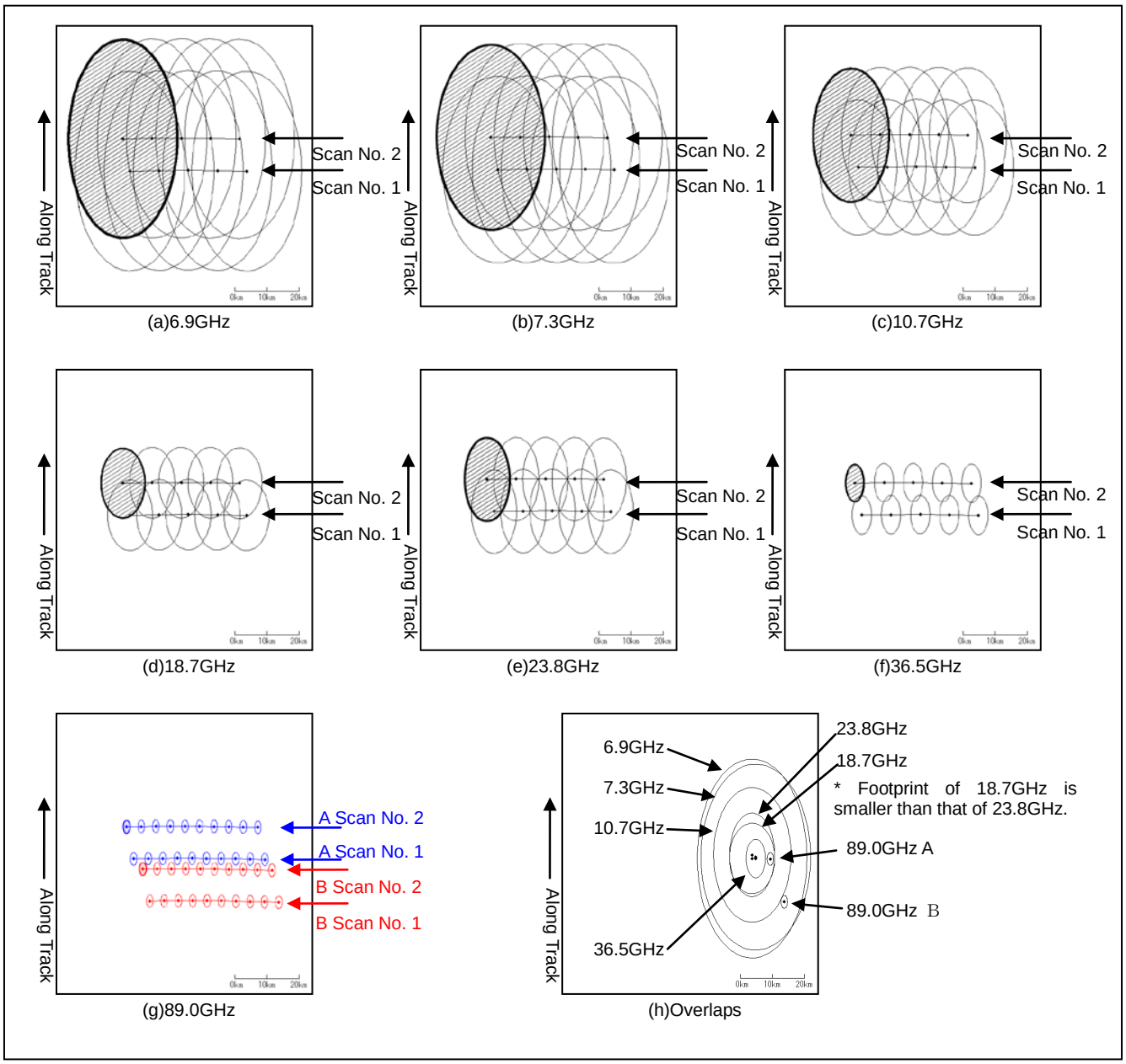


Figure 9. Schematic Image of footprint

Positions of observation point are not strictly same at each frequency channel. Figure 10 shows an example of center latitude and longitude of observation point.

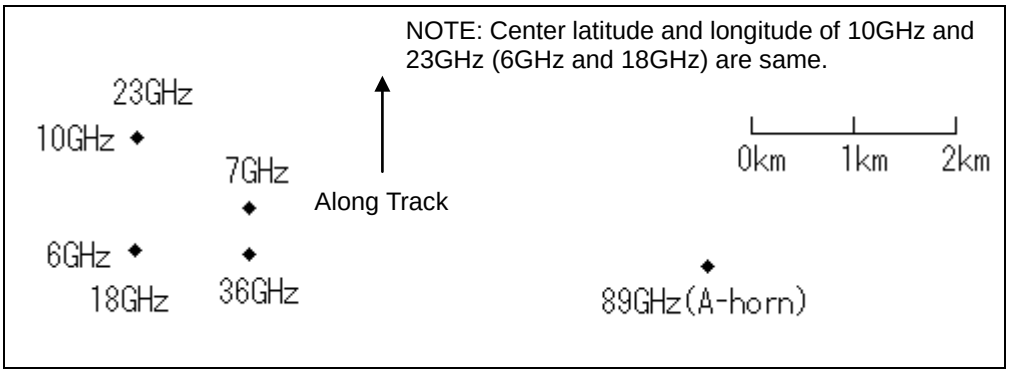


Figure 10. Example of center latitude and longitude of same observation point

3.8 Level1 Resampling Products (L1R)

AMSR2 geophysical parameters are calculated by combining multiple AMSR2 frequencies and polarizations. In that case, there is no consistency with latitude and longitude of same observation point because of different footprint size. AMSR2 provides product call L1R that resamples the L1B product using pre-calculated resampling coefficients to adjust data for lower frequency resolutions.

The center latitude and longitude of data is adjusted to the 89 GHz A channel receiver data.

Resampling converts the brightness temperature at a higher resolution frequency to a brightness temperature at a lower resolution frequency. Resampling data based on 6.9, 10.7, 23.8, 36.5GHz footprint size is created. 6 GHz and 7 GHz data sets have the same spatial resolution; therefore, a shared set of resampling data is created. 18 GHz and 23 GHz data sets have almost the same resolution, but 23 GHz resolution is slightly lower; therefore, one set of resampling data is created matching the 23 GHz resolution. Table 6 shows L1R data and frequencies selected for resampling.

Table 6. L1R data and frequencies selected for resampling

Footprint	6G (H/V) 243 points	7G (H/V) 243 points	10G (H/V) 243 points	18G (H/V) 243 points	23G (H/V) 243 points	36G (H/V) 243 points	89G (H/V) 243 points	89G A-horn (H/V) 486 points	89G B-horn (H/V) 486 points
6GHz	☆	○	○	○	○	○	○		
10GHz			☆	○	○	○	○		
23GHz				○	☆	○	○		
36GHz						☆	○		
89GHz								△	△

○ indicate resampling data undergo spatial resolution conversion and processing for center latitude and longitude alignment.
☆ indicate resampling data only undergo processing for center latitude and longitude alignment.
△ indicate original observational data (same as L1B).
* 18 GHz and 23 GHz data sets have almost the same resolution, but 23 GHz resolution is slightly lower; therefore, one set of resampling data is created matching the 23 GHz resolution.

L1R stores resampling data of 6, 10, 23, 36GHz resolution, and original 89GHz data. L1R data marked by ☆ in Table 6 does not coincident with L1B data of same frequencies and polarizations, because processing for center latitude and longitude alignment is applied.

3.9 Land/Ocean Flag

The land coverage percentage of the observation footprint of AMSR2 is stored for each frequency. Percentage is stored from 0 to 100 % in integer. 0 indicate there is no land in the observation footprint, and 100 indicate that observation footprint is completely covered with lands. Figure 11 shows the image of Land/Ocean Flag values and their footprint.

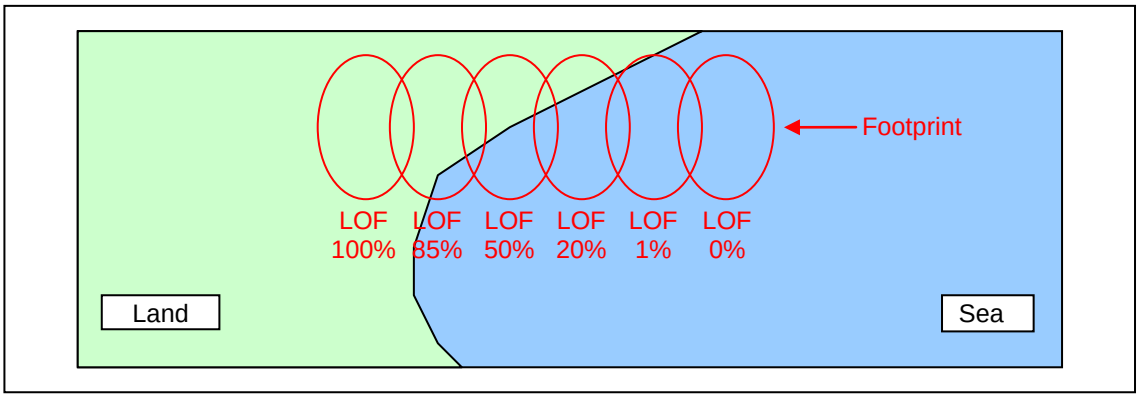


Figure 11. Image of Land/Ocean Flag values and their footprint

Land/Ocean Flag for high frequency and low frequency is stored separately. There are two Land/Ocean Flag data for high frequency channels (89-A, and 89GHz-B).

In the case of L1A and L1B, there are total 6 Land/Ocean Flag data correspond to each low frequency (6, 7, 10, 18, 23, and 36GHz).

In the case of L1R, product stores the Land/Ocean Flag for low frequency channels that are frequencies of 6, 10, 23, 36GHz, because L1R is resampled in 4 kind frequencies as previously mentioned.

3.10 Latitude and Longitude Stored in L1 and L2 Products

Latitude and longitude datat stored in L1 products are that of 89GHz-A and 89GHz-B.

In the case of L1A and L1B, position of 89GHz-A and the co-registration parameters are used for calculating the latitude and longitude of the observing point for each frequency of 6 to 36GHz.

In the case of L1R, processing for center latitude and longitude alignment is applied using the 89 GHz-A data. All data uses position of 89 GHz-A data, but only odd points of 89GHz-A (origin 1) are used for low frequency data.

With AMTK, latitude and longitude of the observation point for each lower frequency channel of L1B and L1R is calculated automatically.

If you only use the HDF5 library in your program codes, L1B need to calculate lower frequency channel's latitude and longitude by yourself. (Subroutine is prepared in sample programs.), L1R need to thin the latitude and longitude of 89GHz-A.

Latitude and longitude data stored in L2 products differs by using either L1B or L1R. (Input L1 data can be checked at L2 metadata of "InputFileName").

In the case of L1B retrieved geophysical parameter, latitude and longitude of same observation point differs by frequency channels. Position stored in L2 is average value of latitude and longitude of each low frequency data.

In the case of L1R retrieved geophysical parameter, latitude and longitude are unified in all frequencies, L2 stores position of low frequency data of L1R.

Precipitation products are high resolution L2 product. Even though these are L1B retrieved geophysical parameter, L2 stores latitude and longitude of 89GHz-A and 89GHz-B.

Latitude and longitude data stored in L1 and L2 is shown in Table 7

Table 7. Latitude and longitude stored in L1 and L2

Product	Stored latitude and longitude	Remarks
L1B	Latitude and longitude of Observation Point for 89A Latitude and longitude of Observation Point for 89B	Position of 89GHz-A and the co-registration parameters are used for calculating the latitude and longitude of the observing point for low frequency channels.
L1R	Latitude and longitude of Observation Point for 89A Latitude and longitude of Observation Point for 89B	Only odd points of 89GHz-A (origin 1) are used for low frequency data.
L2 (L1B Input)	Average value of L1B latitude and longitude of each low frequency channels.	
L2 (L1R Input)	Latitude and longitude of L1R low frequency (Odd points of 89GHz-A)	
L2 (Precipitation)	Latitude and longitude of Observation Point for 89A Latitude and longitude of Observation Point for 89B	

3.11 Altitude Correction Processed in L1R Products

L1B products calculate latitude and longitude as intersection of the earth ellipsoid height and an extended line-of-sight vector. Ground level is not considered. The terrain error (discrepancies between true position and calculated position) showed in Figure 12 is included in the L1B latitude and longitude values caused by the difference between ellipsoid height and the actual ground level.

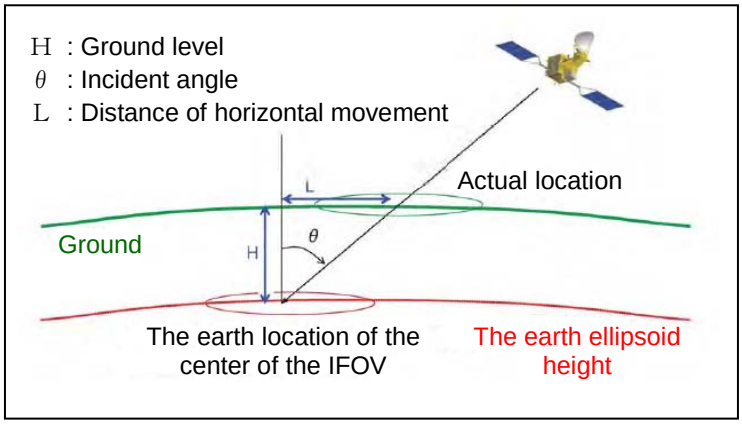


Figure 12. Altitude Correction

L1R stores latitude and longitude which corrected the terrain error of L1B. Distance of horizontal movement L is calculated using ground level H and incident angle θ as follows.

$$L = H\tan\theta$$

3.12 Data Set Specification

Data type, scale factor, and error value of AMSR2 product are shown in Table 8

Table 8. Data type, scale factor, and error value of AMSR2 product

Data	Type	Scale factor[unit]	Error(Missing) value	Outside of observation swath(L3)
L1/L2 scan time (TAI93)	8byte double	1[sec]	-9999.0	
L1/L2 latitude and longitude	4byte float	1[deg]	-9999.0	
L1 incident angle	2byte signed integer	0.01[deg]	-32768	
L1 azimuth angle	2byte signed integer	0.01[deg]	-32768	
L1/L3 brightness temperature	2byte unsigned integer	0.01[K]	65535	65534
L2/L3 geophysical parameters	2byte signed integer	TPW: 0.01[kg/m2] CLW: 0.001[kg/m2] PRC: 0.1[mm/h] SST: 0.01[degC] SSW: 0.01[m/s] SIC: 0.1[%] SND: 0.1[cm] SMC: 0.1[%]	-32768	-32767

To save the size of HDF5 file, 2byte integer is used for some data such as brightness temperatures and geophysical parameters. Value range for 2byte unsigned integer is 0 to 65535, and for 2byte signed integer is -32768 to 32767. Scale factor is set to satisfy the value range of these data types.

For instance, if brightness temperature data is stored as 28312, original value of the data is 283.12[K], since scale factor is set as 0.01[K].

AMTK recognize the scale factor for its conversion and calculate its original value. If you only use the HDF5 library in your program codes, you need to convert the stored value to original value.

Error values are stored in products when there is no geophysical data within observation swath for some reasons. For example, in the case of the geophysical parameters for marine (, such as CLW), the area of land does not compute the geophysical parameters. Then error value of -32768 is stored in land pixel of L2 product.

To distinguish between the area of no geophysical data within observation swath and the area outside of observation swath, L3 products define the former as Missing value which is called Error value in L1/L2 products. The latter is defined as Error value. Figure 13 shows the example image of Missing value and Error value.

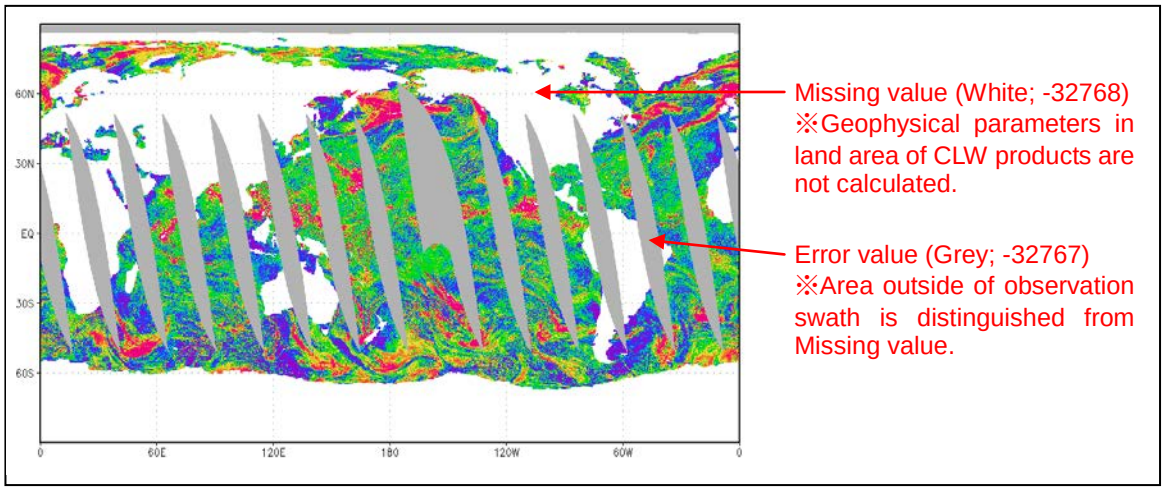


Figure 13. Missing value and error value (CLW)

4. Installation of Libraries

Sample programs used in this document runs under the environment shown in Table 9.

Table 9. Test environment

Hardware	Intel(R) Xeon(R) CPU E5504
OS	Red Hat Enterprise Linux release 5.4
C compiler	GNU C compiler 4.1.2 Intel C compiler Version 11.1 PGI C compiler Version 10.9
FORTRAN compiler	Intel FORTRAN compiler Version 11.1 PGI FORTRAN compiler Version 10.9
HDF5 SZIP	HDF5 version 1.8.4-patch1 SZIP version 2.1
AMTK	version 1.11

In this chapter, we describe how to install the libraries without super user account on Linux machine. Installation directory is set as “/home/user1/util” in this chapter and other directories for each library will be created under installation directory. Directory structure of libraries is shown in Figure 14.

When you install libraries in such directory structure, you can use both old and new version of libraries after version up. And you can also decide which version of libraries to use by switching the library file path.

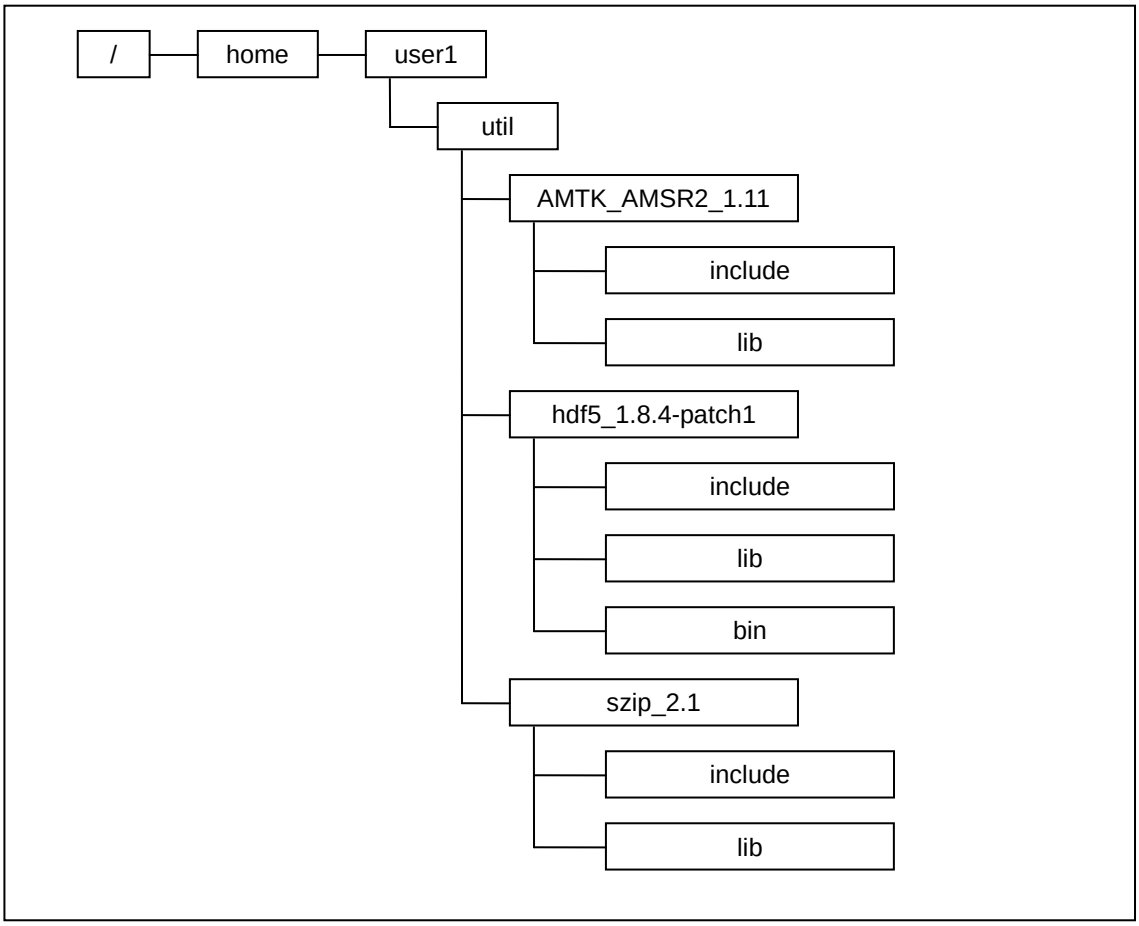


Figure 14. Directory structure

In the following sample, please replace the description of “/home/usr1/util” to your own directory.

(1)Installation of SZIP library

1-1 Download
Get the szip-2.1.tar.gz file (or latest version) from SZIP page of The HDF Group website (http://www.hdfgroup.org/ftp/lib-external/szip/).

1-2 Uncompress
Please type the following commands to uncompress the szip-2.1.tar.gz file.

$ tar -xzvf szip-2.1.tar.gz

If uncompress are completed without error, a new directory “szip-2.1” will be created. Change the current directory to “szip-2.1”.

$ cd szip-2.1

1-3 Compile and installation

Please type the following command in order to compile and install the SZIP library.
SZIP library directory is set to “/home/user1/util/szip_2.1” using “--prefix” option.
* Directory is set as szip_2.1 because the version of SZIP library is 2.1. When the new version released, you must change the directory name to the version you use.

```
$ ./configure --disable-shared --prefix=/home/user1/util/szip_2.1
$ make
$ make install
```

(2)Installation of HDF 5 library

2-1 Download
Get the hdf5-1.8.4-patch1.tar.gz file (or latest version) from The HDF Group website (http://www.hdfgroup.org/).

2-2 Uncompress
Please type the following commands to uncompress the hdf5-1.8.4-patch1.tar.gz file.

$ tar -xzvf hdf5-1.8.4-patch1.tar.gz

If uncompress are completed without error, a new directory “hdf5-1.8.4-patch1” will be created. Change the current directory to “hdf5-1.8.4-patch1”.

$ cd hdf5-1.8.4-patch1

2-3 Compile and installation
Please type the following command in order to compile and install the HDF5 library.
HDF5 library directory is set to “/home/user1/util/hdf5_1.8.4-patch1” by using “--prefix” option.
* Directory is set as hdf5_1.8.4-patch1 because the version of HDF5 library is 1.8.4-patch1. When the new version released, you must change the directory name to the version you use.

＜When you don’t use HDF5 FORTRAN library＞
$ ./configure --disable-shared --prefix=/home/user1/util/hdf5_1.8.4-patch1
--with-szlib=/home/user1/util/szip_2.1
$ make
$ make install

＜When you use HDF5 FORTRAN library＞
$ ./configure --disable-shared --prefix=/home/user1/util/hdf5_1.8.4-patch1
--with-szlib=/home/user1/util/szip_2.1 --enable-fortran FC=ifort
$ make
$ make install

If you only use the HDF5 library in your FORTRAN90 program codes to access the AMSR2 products, please choose the case of ＜When you use HDF5 FORTRAN library＞. FORTRAN compiler is set to “ifort” by using “FC” option. Intel FORTRAN compiler (ifort) and PGI FORTRAN compiler (pgf90) are used for the test environment.

If you don't use AMTK, skip (3) and go to (4).
When you go to (3), process in step (4) is not needed.

(3)Installation of AMTK

3-1 Download
Get the AMTK_AMSR2_Ver1.11.tar.gz file (or latest version) from The GCOM-W1 Data Providing Service (http://gcom-w1.jaxa.jp/).

3-2 Uncompress
Please type the following commands to uncompress the AMTK_AMSR2_Ver1.11.tar.gz file in your installation directory.

$ tar -xzvf AMTK_AMSR2_Ver1.11.tar.gz

If uncompress are completed without error, a new directory "AMTK_AMSR2" will be created. Rename the directory to such name as "AMTK_AMSR2_1.11" which includes the version number. Then change the current directory to "AMTK_AMSR2_1.11"

$ mv AMTK_AMSR2 AMTK_AMSR2_1.11
$ cd AMTK_AMSR2_1.11

Caution）
Directory where you uncompressed tar.gz file and installation directory of SZIP and HDF5 libraries are different.
Installation directory of AMTK should be as same directory as where you uncompressed tar.gz file.

3-3 Compile
Please type the following command in order to compile the AMTK library.

$ ./configure --with-hdf-include=/home/user1/utilhdf5_1.8.4-patch1/include
--with-hdf-lib=/home/user1/utilhdf5_1.8.4-patch1/lib
$ make

Set the directory of included files of the HDF5 library by using "--with-hdf-include" option.
Set the directory of library files of the HDF5 library by using "--with-hdf-lib" option.

3-4 Setting environments
It is necessary to set leap second information file (leapsec.dat) and geophysical quantity definition file (geophysical_file) to environment variable for the execution of the application using the AMTK library. These files are in "share/data" directory which is under the AMTK library directory. Its file name should be written by absolute path. As for the environment parameter, specification is different according to the shell used. Example method of specification for the log in shell is shown below.

<For csh or tcsh>
Please add the following to the ".cshrc" or ".login" file that exists in the home directory.

setenv AMSR2_LEAP_DATA /home/user1/util/AMTK_AMSR2_1.11/share/data/leapsec.dat

setenv GEOPHYSICALFILE /home/user1/util/AMTK_AMSR2_1.11/share/data/geophysical_file

<For sh or bash>
Please add the following to the ".bashrc" or “.profile” file that exists in the home directory.

export AMSR2_LEAP_DATA=/home/user1/util/AMTK_AMSR2_1.11/share/data/leapsec.dat
export GEOPHYSICALFILE=/home/user1/util/AMTK_AMSR2_1.11/share/data/geophysical_file

(4) Setting environment when not using AMTK

Get the leapsec.dat file from The GCOM-W1 Data Providing Service (http://gcom-w1.jaxa.jp/).

Put the file in appropriate directory and set the file name to environment variable AMSR2_LEAP_DATA. In the following sample, we assume that you put leapsec.dat file in the directory “/home/user1/util/”. Example method of specification for the log in shell is shown below.

<For csh or tcsh>
Please add the following to the ".cshrc" or “.login” file that exists in the home directory.

setenv AMSR2_LEAP_DATA /home/user1/util/AMTK_AMSR2_1.11/share/data/leapsec.dat

<For sh or bash>
Please add the following to the ".bashrc" or “.profile” file that exists in the home directory.

export AMSR2_LEAP_DATA=/home/user1/util/AMTK_AMSR2_1.11/share/data/leapsec.dat

5. C Programming with AMTK

Letters written in red are explanation of the sample programs.

Letters written in blue are explanation of the functions used in the sample programs or basic information of the GCOM-W1 satellite and AMSR2 instrument.

5.1 Read L1B Product

5.1.1 C sample program (readL1B_amtk.c)

Sample program (readL1B_amtk.c) which reads the metadata and the datasets of L1B is shown below. Latitude and longitude of low frequency data are also calculated using observation point of 89GHz-A. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans - CoRegistrationParameterA1 - CoRegistrationParameterA2	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B - Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Brightness Temperature (6.9GHz,H) - Brightness Temperature (6.9GHz,V) - Brightness Temperature (7.3GHz,H) - Brightness Temperature (7.3GHz,V) - Brightness Temperature (10.7GHz,H) - Brightness Temperature (10.7GHz,V) - Brightness Temperature (18.7GHz,H) - Brightness Temperature (18.7GHz,V) - Brightness Temperature (23.8GHz,H) - Brightness Temperature (23.8GHz,V) - Brightness Temperature (36.5GHz,H) - Brightness Temperature (36.5GHz,V) - Brightness Temperature (89.0GHz-A,H) - Brightness Temperature (89.0GHz-A,V) - Brightness Temperature (89.0GHz-B,H) - Brightness Temperature (89.0GHz-B,V) * Pixel Data Quality 6 to 36 - Pixel Data Quality 89 * Land_Ocean Flag 6 to 36 - Land_Ocean Flag 89 * Earth Incidence - Earth Azimuth
Data calculated from observation point of 89GHz-A	
- Latitude and longitude (Low mean) - Latitude and longitude (6G) - Latitude and longitude (7G) - Latitude and longitude (10G) - Latitude and longitude (18G) - Latitude and longitude (23G) - Latitude and longitude (36G) For details to P.20	

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.

For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure “AM2_COMMON_SCANTIME” -> AMTK_getScanTime function

- Output data type is data structure “AM2_COMMON_LATLON” -> AMTK_getLatLon function
- Output data type is float -> AMTK_get_SwathFloat function
- Output data type is integer -> AMTK_get_SwathInt function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "AMTK.h"
 4
 5 // fixed value
 6 #define LMT 2200 // limit of NumberOfScans
 7
 8 int main(int argc, char *argv[]){
 9   // interface variable
10   int i,j;      // loop variable
11   int ret;      // return status
12   char buf[512]; // text buffer
13   void *vpnt;    // pointer to void
14   char *fn;     // filename
15   hid_t hnd;    // file handle
16
17   // meta data
18   char geo[512];  // GeophysicalName
19   char gid[512];  // GranuleID
20   char tm1[512];  // ObservationStartDateTime
21   char tm2[512];  // EquatorCrossingDateTime
22   char tm3[512];  // ObservationEndDateTime
23   int num;       // NumberOfScans
24   int ovr;       // OverlapScans
25   char prm1[512]; // CoRegistrationParameterA1
26   char prm2[512]; // CoRegistrationParameterA2
 :
28   // array dat
29   AM2_COMMON_SCANTIME st[LMT]; // scantime
30   AM2_COMMON_LATLON ll89a[LMT][AM2_DEF_SNUM_HI]; // latlon for 89a
31   AM2_COMMON_LATLON ll89b[LMT][AM2_DEF_SNUM_HI]; // latlon for 89b
 :
39   float tb06h [LMT][AM2_DEF_SNUM_LO]; // tb for 06h
40   float tb06v [LMT][AM2_DEF_SNUM_LO]; // tb for 06v
 :
56   unsigned char pdq06h [LMT][AM2_DEF_SNUM_LO];    // pixel data quality for 06h
57   unsigned char pdq06v [LMT][AM2_DEF_SNUM_LO];    // pixel data quality for 06v
 :
66   unsigned char lof06 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 06
67   unsigned char lof07 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 07

75   float ear_in[AM2_DEF_SNUM_LO][LMT]; // earth incidence
76   float ear_az[AM2_DEF_SNUM_LO][LMT]; // earth azimuth
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
 88   // open
 89   hnd=AMTK_openH5(fn);
 90   if(hnd<0){
 91     printf("AMTK_openH5 error: %s¥n", fn);
 92     printf("amtk status: %d¥n", hnd);
 93     exit(1);
 94   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata。

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
 96   // read meta: GeophysicalName
 97   vpnt=geo;
 98   ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
 99   if(ret<0){
100     printf("AMTK_getMetaDataName error: GeophysicalName¥n");
101     printf("amtk status: %d¥n",ret);
102     exit(1);
103   }
104   printf("GeophysicalName: %s¥n",geo);
  :
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
146   // read meta: NumberOfScans
147   vpnt=buf;
148   ret=AMTK_getMetaDataName(hnd,"NumberOfScans",(char **)&vpnt);
149   if(ret<0){
150     printf("AMTK_getMetaDataName error: NumberOfScans¥n");
151     printf("amtk status: %d¥n",ret);
152     exit(1);
153   }
154   num=atoi(buf);
155   printf("NumberOfScans: %d¥n",num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

For details to P.14

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
197   // read array: scantime
198   vpnt=st;
199   ret=AMTK_getScanTime(hnd,1,num,(AM2_COMMON_SCANTIME **)&vpnt);
200   if(ret<0){
201     printf("AMTK_getScanTime error.¥n");
202     printf("amtk status: %d¥n",ret);
203     exit(1);
204   }
205   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
206   , st[0].year
207   , st[0].month
208   , st[0].day
209   , st[0].hour
210   , st[0].minute
211   , st[0].second
212   );
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20

Read latitude and longitude.
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_89A (access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
214   // read array: latlon for 89a
215   vpnt=ll89a;
216   ret=AMTK_getLatLon(hnd,(AM2_COMMON_LATLON **)&vpnt,1,num,AM2_LATLON_89A);
217   if(ret<0){
218     printf("AMTK_getLatLon error: AM2_LATLON_89A¥n");
219     printf("amtk status: %d¥n",ret);
220     exit(1);
221   }
222   printf("latlon89a[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", ll89a[0][0].lat,
ll89a[0][0].lon);
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of brightness temperature data.
Brightness temperature data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_TB06H (access label for “Brightness Temperature (6.9GHz,H)”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
304   // read array: tb for 06h
305   vpnt=tb06h;
306   ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_TB06H);
307   if(ret<0){
308     printf("AMTK_get_SwathFloat error: AM2_TB06H¥n");
309     printf("amtk status: %d¥n",ret);
310     exit(1);
311   }
312   printf("tb06h[scan=0][pixel=0]: %9.2f¥n", tb06h[0][0]);
```

Read L1B pixel data quality * Numbers written on the left are row number of the sample program.

Read L1B pixel data quality
Function AMTK_get_SwathInt and int type array is used for the acquisition of pixel data quality information.
Pixel data quality information is two dimensional (“Pixel Data Quality 6 to 36”:(sample * 16) * scan, “Pixel Data Quality 89”:(sample * 8) * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathInt(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Int type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_LO (access label for “Pixel Data Quality 6 to 36”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
464   // read array: pixel data quality for low
465   vpnt=pdqlo;
466   ret=AMTK_get_SwathInt(hnd,(int **)&vpnt,1,num,AM2_PIX_QUAL_LO);
467   if(ret<0){
468     printf("AMTK_get_SwathInt error: AM2_PIX_QUAL_LO¥n");
469     printf("amtk status: %d¥n",ret);
470     exit(1);
471   }
```

“Pixel Data Quality 6 to 36” are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to unsigned char type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
471   for(j=0;j<num;++j){
472     for(i=0;i<AM2_DEF_SNUM_LO;++i){
473       pdq06v[j][i]=pdqlo[j][i*16+ 1]*10+pdqlo[j][i*16+ 0];
474       pdq06h[j][i]=pdqlo[j][i*16+ 3]*10+pdqlo[j][i*16+ 2];
475       pdq07v[j][i]=pdqlo[j][i*16+ 5]*10+pdqlo[j][i*16+ 4];
476       pdq07h[j][i]=pdqlo[j][i*16+ 7]*10+pdqlo[j][i*16+ 6];
477     }
478   }
```

Read L1B land ocean flag * Numbers written on the left are row number of the sample program.

Read L1B land ocean flag.
Function AMTK_get_SwathInt and int type array is used for the acquisition of land ocean flag data.
Land ocean flag data is two dimensional (“Land/Ocean Flag 6 to 36”: sample * (scan * 6), “Land/Ocean Flag 89”: sample * (scan * 2)), and data from 1 scan to “num” scan is read in the sample program.

For details to P.18

```
506   // read array: land ocean flag for low
507   vpnt=loflo;
508   ret=AMTK_get_SwathInt(hnd,(int **)&vpnt,1,num,AM2_LOF_LO);
509   if(ret<0){
510     printf("AMTK_get_SwathInt error: AM2_LOF_LO¥n");
511     printf("amtk status: %d¥n",ret);
512     exit(1);
513   }
```

“Land/Ocean Flag 6 to 36” and “Land/Ocean Flag 89” is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
514   for(j=0;j<num;++j){
515     for(i=0;i<AM2_DEF_SNUM_LO;++i){
516       lof06[j][i]=loflo[num*0+j][i];
517       lof07[j][i]=loflo[num*1+j][i];
518       lof10[j][i]=loflo[num*2+j][i];
519       lof18[j][i]=loflo[num*3+j][i];
520       lof23[j][i]=loflo[num*4+j][i];
521       lof36[j][i]=loflo[num*5+j][i];
522     }
523   }
```

Read earth incidence * Numbers written on the left are row number of the sample program.

Read earth incidence.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of earth incidence data.
Earth incidence data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

```
548   // read array: earth incidence
549   vpnt=ear_in;
550   ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_EARTH_INC);
551   if(ret<0){
552     printf("AMTK_get_SwathFloat error: AM2_EARTH_INC¥n");
553     printf("amtk status: %d¥n",ret);
554     exit(1);
555   }
556   printf("ear_in[scan=0][pixel=0]: %9.2f¥n",ear_in[0][0]);
```

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
568   // close
569   ret=AMTK_closeH5(hnd);
```

5.1.2 Compile (Explanation of build_readL1B_amtk_c.sh)

We explain how to compile the C program by using script “build_readL1B_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL1B_amtk.c
16
17 # output filename
18 out=readL1B_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL1B_amtk_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1B_amtk_c.sh
icc -g readL1B_amtk.c -o readL1B_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.1.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >
$ unlimit

＜For sh or bash＞
* Type the following four commands in order.
$ ulimit -d unlimited
$ ulimit -m unlimited
$ ulimit -s unlimited
$ ulimit -v unlimited

The example of executing “readL1B_amtk_c” is shown as follows.

```
$ ./readL1B_amtk_c GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGBTBR_0000000
ObservationStartDateTime: 2012-07-26T11:45:43.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:37.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G-1.25000,7G-1.00000,10G-1.25000,18G-1.25000,23G-1.25000,36G-1.000000
CoRegistrationParameterA2: 6G-0.00000,7G--0.10000,10G--0.25000,18G-0.00000,23G--0.25000,36G-0.00000
time[scan=0]: 2012/07/26 11:45:43
latlon89a[scan=0][pixel=0]: ( -73.3289, 136.7714)
latlon89b[scan=0][pixel=0]: ( -73.4038, 137.1498)
latlonlm[scan=0][pixel=0]: ( -73.3538, 136.6228)
latlon06[scan=0][pixel=0]: ( -73.3592, 136.6213)
latlon07[scan=0][pixel=0]: ( -73.3497, 136.6429)
latlon10[scan=0][pixel=0]: ( -73.3506, 136.6001)
latlon18[scan=0][pixel=0]: ( -73.3592, 136.6213)
latlon23[scan=0][pixel=0]: ( -73.3506, 136.6001)
latlon36[scan=0][pixel=0]: ( -73.3532, 136.6514)
tb06h[scan=0][pixel=0]:   173.28
tb06v[scan=0][pixel=0]:   208.22
tb07h[scan=0][pixel=0]:   173.07
tb07v[scan=0][pixel=0]:   207.54
tb10h[scan=0][pixel=0]:   170.94
tb10v[scan=0][pixel=0]:   204.95
tb18h[scan=0][pixel=0]:   164.85
tb18v[scan=0][pixel=0]:   199.84
tb23h[scan=0][pixel=0]:   163.22
tb23v[scan=0][pixel=0]:   196.53
tb36h[scan=0][pixel=0]:   156.56
tb36v[scan=0][pixel=0]:   186.39
tb89ah[scan=0][pixel=0]:   163.76
tb89av[scan=0][pixel=0]:   179.27
tb89bh[scan=0][pixel=0]:   170.60
tb89bv[scan=0][pixel=0]:   188.16
pdq06h[scan=0][pixel=0]):   0
pdq06v[scan=0][pixel=0]):   0
pdq07h[scan=0][pixel=0]):   0
pdq07v[scan=0][pixel=0]):   0
pdq10h[scan=0][pixel=0]):   0
pdq10v[scan=0][pixel=0]):   0
pdq18h[scan=0][pixel=0]):   0
pdq18v[scan=0][pixel=0]):   0
pdq23h[scan=0][pixel=0]):   0
pdq23v[scan=0][pixel=0]):   0
pdq36h[scan=0][pixel=0]):   0
pdq36v[scan=0][pixel=0]):   0
pdq89ah[scan=0][pixel=0]):   0
pdq89av[scan=0][pixel=0]):   0
pdq89ah[scan=0][pixel=0]):   0
pdq89av[scan=0][pixel=0]):   0
lof06[scan=0][pixel=0]: 100
lof07[scan=0][pixel=0]: 100
lof10[scan=0][pixel=0]: 100
lof18[scan=0][pixel=0]: 100
lof23[scan=0][pixel=0]: 100
lof36[scan=0][pixel=0]: 100
lof89a[scan=0][pixel=0]: 100
lof89b[scan=0][pixel=0]: 100
ear_in[scan=0][pixel=0]:    55.20
ear_az[scan=0][pixel=0]:   144.76
```

5.2 Read L1R Product

5.2.1 C sample program (readL1R_amtk.c)

Sample program (readL1R_amtk.c) which reads the metadata and the datasets of L1R is shown below. Latitude and longitude of low frequency data are extracted from observation point of 89GHz-A. Values of data are dumped to the standard output.

Only part of L1R brightness temperature data are handled in this sample program. Please refer to "3.8 Level1 Resampling Products (L1R)" on page 18.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans - CoRegistrationParameterA1 - CoRegistrationParameterA2	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B - Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Brightness Temperature (res06,6.9GHz,H) - Brightness Temperature (res06,6.9GHz,V) - Brightness Temperature (res06,7.3GHz,H) - Brightness Temperature (res06,7.3GHz,V) - Brightness Temperature (res10,10.7GHz,H) - Brightness Temperature (res10,10.7GHz,V) - Brightness Temperature (res23,18.7GHz,H) - Brightness Temperature (res23,18.7GHz,V) - Brightness Temperature (res23,23.8GHz,H) - Brightness Temperature (res23,23.8GHz,V) - Brightness Temperature (res36,36.5GHz,H) - Brightness Temperature (res36,36.5GHz,V) - Brightness Temperature (res36,89.0GHz,H) - Brightness Temperature (res36,89.0GHz,V) - Brightness Temperature (original,89GHz-A,H) - Brightness Temperature (original,89GHz-A,V) - Brightness Temperature (original,89GHz-B,H) - Brightness Temperature (original,89GHz-B,V) * Pixel Data Quality 6 to 36 - Pixel Data Quality 89 * Land_Ocean Flag 6 to 36 - Land_Ocean Flag 89 * Earth Incidence - Earth Azimuth
Data extracted from observation point of 89GHz-A	
- Latitude and longitude of low frequency data For details to P.20	

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.

For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure "AM2_COMMON_SCANTIME" -> AMTK_getScanTime function
- Output data type is data structure "AM2_COMMON_LATLON" -> AMTK_getLatLon function

- Output data type is float -> AMTK_get_SwathFloat function
- Output data type is integer -> AMTK_get_SwathInt function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "AMTK.h"
 4
 5 // fixed value
 6 #define LMT 2200 // limit of NumberOfScans
 7
 8 int main(int argc, char *argv[]){
 9   // interface variable
10   int i,j;      // loop variable
11   int ret;      // return status
12   char buf[512]; // text buffer
13   void *vpnt;   // pointer to void
14   char *fn;     // filename
15   hid_t hnd;    // file handle
16
17   // meta data
18   char geo[512];  // GeophysicalName
19   char gid[512];  // GranuleID
20   char tm1[512];  // ObservationStartDateTime
21   char tm2[512];  // EquatorCrossingDateTime
22   char tm3[512];  // ObservationEndDateTime
23   int num;       // NumberOfScans
24   int ovr;       // OverlapScans
25   char prm1[512]; // CoRegistrationParameterA1
26   char prm2[512]; // CoRegistrationParameterA2

28   // array data
29   AM2_COMMON_SCANTIME st[LMT]; // scantime
30   AM2_COMMON_LATLON ll89ar[LMT][AM2_DEF_SNUM_HI]; // latlon for 89a altitude
revised
31   AM2_COMMON_LATLON ll89br[LMT][AM2_DEF_SNUM_HI]; // latlon for 89b altitude
revised
 :
33   float tb06h06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06h, resolution 06G
34   float tb06v06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06v, resolution 06G
 :
52   unsigned char pdq06h [LMT][AM2_DEF_SNUM_LO];    // pixel data quality for 06h
53   unsigned char pdq06v [LMT][AM2_DEF_SNUM_LO];    // pixel data quality for 06v
 :
62   unsigned char lof06 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 06
63   unsigned char lof10 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 10
 :
69   float ear_in[AM2_DEF_SNUM_LO][LMT]; // earth incidence
70   float ear_az[AM2_DEF_SNUM_LO][LMT]; // earth azimuth
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name
hnd: [Return value]Normal: HDF access file id is returned. Error: A negative value is returned.

```
 82   // open
 83   hnd=AMTK_openH5(fn);
 84   if(hnd<0){
 85     printf("AMTK_openH5 error: %s¥n", fn);
 86     printf("amtk status: %d¥n", hnd);
 87     exit(1);
 88   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.S
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
 90   // read meta: GeophysicalName
 91   vpnt=geo;
 92   ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
 93   if(ret<0){
 94     printf("AMTK_getMetaDataName error: GeophysicalName¥n");
 95     printf("amtk status: %d¥n",ret);
 96     exit(1);
 97   }
 98   printf("GeophysicalName: %s¥n",geo);
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
140   // read meta: NumberOfScans
141   vpnt=buf;
142   ret=AMTK_getMetaDataName(hnd,"NumberOfScans",(char **)&vpnt);
143   if(ret<0){
144     printf("AMTK_getMetaDataName error: NumberOfScans¥n");
145     printf("amtk status: %d¥n",ret);
146     exit(1);
147   }
148   num=atoi(buf);
149   printf("NumberOfScans: %d¥n",num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

For details to P.14.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
191   // read array: scantime
192   vpnt=st;
193   ret=AMTK_getScanTime(hnd,1,num,(AM2_COMMON_SCANTIME **)&vpnt);
194   if(ret<0){
195     printf("AMTK_getScanTime error.¥n");
196     printf("amtk status: %d¥n",ret);
197     exit(1);
198   }
199   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
200   , st[0].year
201   , st[0].month
202   , st[0].day
203   , st[0].hour
204   , st[0].minute
205   , st[0].second
206   );
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20.

Read latitude and longitude
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_RS_89A (access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
208   // read array: latlon for 89a altitude revised
209   vpnt=ll89ar;
210   ret=AMTK_getLatLon(hnd,(AM2_COMMON_LATLON **)&vpnt,1,num
,AM2_LATLON_RS_89A);
211   if(ret<0){
212     printf("AMTK_getLatLon error: AM2_LATLON_RS_89A¥n");
213     printf("amtk status: %d¥n",ret);
214     exit(1);
215   }
216   printf("latlon89ar[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", ll89ar[0][0].lat,
ll89ar[0][0].lon);
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of brightness temperature data.
Brightness temperature data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_TB06H (access label for “Brightness Temperature (6.9GHz,H)”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
238   // read array: tb for 06h, resolution 06G
239   vpnt=tb06h06;
240   ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_RES06_TB06H);
241   if(ret<0){
242     printf("AMTK_get_SwathFloat error: AM2_RES06_TB06H¥n");
243     printf("amtk status: %d¥n",ret);
244     exit(1);
245   }
246   printf("tb06h06[scan=0][pixel=0]: %9.2f¥n", tb06h06[0][0]);
```

Read L1R pixel data quality * Numbers written on the left are row number of the sample program.

Read L1R pixel data quality
Function AMTK_get_SwathInt and int type array is used for the acquisition of pixel data quality information.
Pixel data quality information is two dimensional (“Pixel Data Quality 6 to 36”:(sample * 16) * scan, “Pixel Data Quality 89”:(sample * 8) * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathInt(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Int type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_LO (access label for “Pixel Data Quality 6 to 36”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
418   // read array: pixel data quality for low
419   vpnt=pdqlo;
420   ret=AMTK_get_SwathInt(hnd,(int **)&vpnt,1,num,AM2_PIX_QUAL_LO);
421   if(ret<0){
422     printf("AMTK_get_SwathInt error: AM2_PIX_QUAL_LO¥n");
423     printf("amtk status: %d¥n",ret);
424     exit(1);
425   }
```

“Pixel Data Quality 6 to 36” are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to unsigned char type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
426   for(j=0;j<num;++j){
427     for(i=0;i<AM2_DEF_SNUM_LO;++i){
428       pdq06v[j][i]=pdqlo[j][i*16+ 1]*10+pdqlo[j][i*16+ 0];
429       pdq06h[j][i]=pdqlo[j][i*16+ 3]*10+pdqlo[j][i*16+ 2];
430       pdq07v[j][i]=pdqlo[j][i*16+ 5]*10+pdqlo[j][i*16+ 4];
431       pdq07h[j][i]=pdqlo[j][i*16+ 7]*10+pdqlo[j][i*16+ 6];
432     }
433   }
```

Read L1R land ocean flag * Numbers written on the left are row number of the sample program.

Read L1R land ocean flag.
Function AMTK_get_SwathInt and int type array is used for the acquisition of land ocean flag data.
Land ocean flag data is two dimensional ("Land/Ocean Flag 6 to 36": sample * (scan * 4), "Land/Ocean Flag 89": sample * (scan*2)), and data from 1 scan to "num" scan is read in the sample program.

For details to P.18.

```
460   // read array: land ocean flag for low
461   vpnt=loflo;
462   ret=AMTK_get_SwathInt(hnd,(int **)&vpnt,1,num,AM2_LOF_RES_LO);
463   if(ret<0){
464     printf("AMTK_get_SwathInt error: AM2_LOF_RES_LO¥n");
465     printf("amtk status: %d¥n",ret);
466     exit(1);
467   }
```

"Land/Ocean Flag 6 to 36" and "Land/Ocean Flag 89"is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
468   for(j=0;j<num;++j){
469     for(i=0;i<AM2_DEF_SNUM_LO;++i){
470       lof06[j][i]=loflo[num*0+j][i];
471       lof10[j][i]=loflo[num*1+j][i];
472       lof23[j][i]=loflo[num*2+j][i];
473       lof36[j][i]=loflo[num*3+j][i];
474     }
475   }
```

Read earth incidence * Numbers written on the left are row number of the sample program.

Read earth incidence.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of earth incidence data.
Earth incidence data is two dimensional (sample * scan), and data from 1 scan to "num" scan is read in the sample program.

```
498   // read array: earth incidence
499   vpnt=ear_in;
500   ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_EARTH_INC);
501   if(ret<0){
502     printf("AMTK_get_SwathFloat error: AM2_EARTH_INC¥n");
503     printf("amtk status: %d¥n",ret);
504     exit(1);
505   }
506   printf("ear_in[scan=0][pixel=0]: %9.2f¥n",ear_in[0][0]);
```

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
518   // close
519   ret=AMTK_closeH5(hnd);
```

5.2.2 Compile (Explanation of build_readL1R_amtk_c.sh)

We explain how to compile the C program by using script “build_readL1R_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL1R_amtk.c
16
17 # output filename
18 out=readL1R_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL1R_amtk_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1R_amtk_c.sh
icc -g readL1R_amtk.c -o readL1R_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.2.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of “executing readL1R_amtk_c” is shown as follows.

```
$ ./readL1R_amtk_c GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGRTBR_000000
ObservationStartDateTime: 2012-07-26T11:45:43.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:37.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G-0.00000,7G-0.00000,10G-0.00000,18G-0.00000,23G-0.00000,36G-0.000000
CoRegistrationParameterA2: 6G-0.00000,7G-0.00000,10G-0.00000,18G-0.00000,23G-0.00000,36G-0.000000
time[scan=0]: 2012/07/26 11:45:43
latlon89ar[scan=0][pixel=0]: ( -73.3581, 136.8432)
latlon89br[scan=0][pixel=0]: ( -73.4328, 137.2216)
latlonlr[scan=0][pixel=0]: ( -73.3581, 136.8432)
tb06h06[scan=0][pixel=0]:   173.41
tb06v06[scan=0][pixel=0]:   208.09
tb07h06[scan=0][pixel=0]:   173.07
tb07v06[scan=0][pixel=0]:   207.11
tb10h10[scan=0][pixel=0]:   170.40
tb10v10[scan=0][pixel=0]:   204.58
tb18h23[scan=0][pixel=0]:   165.83
tb18v23[scan=0][pixel=0]:   199.41
tb23h23[scan=0][pixel=0]:   163.55
tb23v23[scan=0][pixel=0]:   195.90
tb36h36[scan=0][pixel=0]:   153.55
tb36v36[scan=0][pixel=0]:   183.95
tb89h36[scan=0][pixel=0]:   163.86
tb89v36[scan=0][pixel=0]:   181.05
tb89ah[scan=0][pixel=0]:   163.76
tb89av[scan=0][pixel=0]:   179.27
tb89bh[scan=0][pixel=0]:   170.60
tb89bv[scan=0][pixel=0]:   188.16
pdq06h[scan=0][pixel=0]:   0
pdq06v[scan=0][pixel=0]:   0
pdq07h[scan=0][pixel=0]:   0
pdq07v[scan=0][pixel=0]:   0
pdq10h[scan=0][pixel=0]:   0
pdq10v[scan=0][pixel=0]:   0
pdq18h[scan=0][pixel=0]:   0
pdq18v[scan=0][pixel=0]:   0
pdq23h[scan=0][pixel=0]:   0
pdq23v[scan=0][pixel=0]:   0
pdq36h[scan=0][pixel=0]:   0
pdq36v[scan=0][pixel=0]:   0
pdq89ah[scan=0][pixel=0]:   0
pdq89av[scan=0][pixel=0]:   0
pdq89ah[scan=0][pixel=0]:   0
pdq89av[scan=0][pixel=0]:   0
lof06[scan=0][pixel=0]: 100
lof10[scan=0][pixel=0]: 100
lof23[scan=0][pixel=0]: 100
lof36[scan=0][pixel=0]: 100
lof89a[scan=0][pixel=0]: 100
lof89b[scan=0][pixel=0]: 100
ear_in[scan=0][pixel=0]:    55.20
ear_az[scan=0][pixel=0]:   144.76
```

5.3 Read L2 Low Resolution Product

L2 Low Resolution Products are Cloud Liquid Water (CLW), Sea Ice Concentration (SIC), Soil Moisture Content (SMC), Snow Depth (SND), Sea Surface Temperature (SST), Sea Surface Wind Speed (SSW), and Total Precipitable Water (TPW). For Precipitation (PRC) product, please refer to "5.4 Read L2 High Resolution Product" on page 60.

5.3.1 C sample program (readL2L_amtk.c)

Sample program (readL2L_amtk.c) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point * Longitude of Observation Point * Geophysical Data * Pixel Data Quality For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.

For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure "AM2_COMMON_SCANTIME" -> AMTK_getScanTime function
- Output data type is data structure "AM2_COMMON_LATLON" -> AMTK_getLatLon function
- Output data type is float -> AMTK_get_SwathFloat function
- Output data type is unsigned char -> AMTK_get_SwathUchar function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "AMTK.h"
 5
 6 // fixed value
 7 #define LMT 2200 // limit of NumberOfScans
 :
10 int main(int argc, char *argv[]){
11   // interface variable
12   int i,j;      // loop variable
13   int ret;      // return status
14   char buf[512]; // text buffer
15   void *vpnt;   // pointer to void
16   char *fn;     // filename
17   hid_t hnd;    // file handle
18
19   // meta data
20   char geo[512];  // GeophysicalName
21   char gid[512];  // GranuleID
22   char tm1[512];  // ObservationStartDateTime
23   char tm2[512];  // EquatorCrossingDateTime
24   char tm3[512];  // ObservationEndDateTime
25   int num;       // NumberOfScans
26   int ovr;       // OverlapScans


28   // array data
29   AM2_COMMON_SCANTIME st[LMT]; // scantime
30   AM2_COMMON_LATLON ll[LMT][AM2_DEF_SNUM_LO]; // latlon

31   float geo1[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 1
32   float geo2[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 2

33   unsigned char pdq1[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 1
34   unsigned char pdq2[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 2
35   unsigned char pdqtmp[LMT*2][AM2_DEF_SNUM_LO]; // pixel data quality temporary
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
47   // open
48   hnd=AMTK_openH5(fn);
49   if(hnd<0){
50     printf("AMTK_openH5 error: %s¥n", fn);
51     printf("amtk status: %d¥n", hnd);
52     exit(1);
53   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
55   // read meta: GeophysicalName
56   vpnt=geo;
57   ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
58   if(ret<0){
59     printf("AMTK_getMetaDataName error: GeophysicalName¥n");
60     printf("amtk status: %d¥n",ret);
61     exit(1);
62   }
63   printf("GeophysicalName: %s¥n",geo);
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
118  // read meta: NumberOfScans
119  vpnt=buf;
120  ret=AMTK_getMetaDataName(hnd,"NumberOfScans",(char **)&vpnt);
121  if(ret<0){
122    printf("AMTK_getMetaDataName error: NumberOfScans¥n");
123    printf("amtk status: %d¥n",ret);
124    exit(1);
125  }
126  num=atoi(buf);
127  printf("NumberOfScans: %d¥n",num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

For details to P.14.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
149   // read array: scantime
150   vpnt=st;
151   ret=AMTK_getScanTime(hnd,1,num,(AM2_COMMON_SCANTIME **)&vpnt);
152   if(ret<0){
153     printf("AMTK_getScanTime error.¥n");
154     printf("amtk status: %d¥n",ret);
155     exit(1);
156   }
157   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
158   , st[0].year
159   , st[0].month
160   , st[0].day
161   , st[0].hour
162   , st[0].minute
163   , st[0].second
164   );
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20.

Read latitude and longitude.
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_L2_LO (access label for “Latitude of Observation Point” and “Longitude of Observation Point”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
166   // read array: latlon
167   vpnt=ll;
168   ret=AMTK_getLatLon(hnd,(AM2_COMMON_LATLON **)&vpnt,1,num
,AM2_LATLON_L2_LO);
169   if(ret<0){
170     printf("AMTK_getLatLon error: AM2_LATLON_L2_LO¥n");
171     printf("amtk status: %d¥n",ret);
172     exit(1);
173   }
174   printf("latlon[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", ll[0][0].lat,
ll[0][0].lon);
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of geophysical quantity data.
Geophysical quantity data is three dimensional (layer * sample * scan) and data from 1 scan to "num" scan is read in the sample program.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is "Snow Water Equivalent".
Geophysical quantity stored in the second layer of sea surface temperature product is "SST obtained by 10GHz".

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_SWATH_GEO1 (access label for the first layer of "Geophysical Data") and AM2_SWATH_GEO2 (access label for the second layer of "Geophysical Data") is shown below.
ret: [Return value] Error: A negative value is returned.

```
176   // read array: geophysical data for 1 layer
177   if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
178     vpnt=geo1;
179     ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_SWATH_GEO1);
180     if(ret<0){
181       printf("AMTK_get_SwathFloat error: AM2_SWATH_GEO1¥n");
182       printf("amtk status: %d¥n",ret);
183       exit(1);
184     }
185   }
186
```

You can read the second layer by changing the access label.

```
187   // read array: geophysical data for 2 layer
188   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
189     // layer 1
190     vpnt=geo1;
191     ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_SWATH_GEO1);
192     if(ret<0){
193       printf("AMTK_get_SwathFloat error: AM2_SWATH_GEO1¥n");
194       printf("amtk status: %d¥n",ret);
195       exit(1);
196     }
197     // layer 2
198     vpnt=geo2;
199     ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_SWATH_GEO2);
200     if(ret<0){
201       printf("AMTK_get_SwathFloat error: AM2_SWATH_GEO2¥n");
202       printf("amtk status: %d¥n",ret);
203       exit(1);
204     }
205   }
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.
Function AMTK_get_SwathUchar and unsigned char type array is used for the acquisition of L2 pixel data quality information.
L2 pixel data quality information is three dimensional (sample * scan * layer), and data from 1 scan to “num” scan is read in the sample program.
Snow depth product and Sea surface temperature product have two layer of pixel data quality.

ret=AMTK_get_SwathUchar(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Unsigned char type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL (access label for “Pixel Data Quality”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
207   // read array: pixel data quality for 1 layer
208   if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
209     vpnt=pdq1;
210     ret=AMTK_get_SwathUChar(hnd,(unsigned char **)&vpnt,1,num,AM2_PIX_QUAL);
211     if(ret<0){
212       printf("AMTK_get_SwathUChar error: AM2_PIX_QUAL¥n");
213       printf("amtk status: %d¥n",ret);
214       exit(1);
215     }
216   }
217
```

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
218   // read array: pixel data quality for 2 layer
219   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
220     // read
221     vpnt=pdqtmp;
222     ret=AMTK_get_SwathUChar(hnd,(unsigned char **)&vpnt,1,num,AM2_PIX_QUAL);
223     if(ret<0){
224       printf("AMTK_get_SwathUChar error: AM2_PIX_QUAL¥n");
225       printf("amtk status: %d¥n",ret);
226       exit(1);
227     }
228     // separate
229     for(j=0;j<num;++j){
230       for(i=0;i<AM2_DEF_SNUM_LO;++i){
231         pdq1[j][i]=pdqtmp[num*0+j][i];
232         pdq2[j][i]=pdqtmp[num*1+j][i];
233       }
234     }
235   }
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
261   // close
262   ret=AMTK_closeH5(hnd);
```

5.3.2 Compile (Explanation of build_readL2L_amtk_c.sh)

We explain how to compile the C program by using script “build_readL2L_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL2L_amtk.c
16
17 # output filename
18 out=readL2L_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL2L_amtk_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2L_amtk_c.sh
icc -g readL2L_amtk.c -o readL2L_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.3.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2L_amtk_c” is shown as follows.

```
$ ./readL2L_amtk_c GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_201303011809_125D_L2SGCLWLA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans: 1972
limit of NumberOfScans = 2200
OverlapScans: 0
time[scan=0]: 2013/03/01 18:09:10
latlon[scan=0][pixel=0]: (    84.4574, -78.1076)
geo1[scan=0][pixel=0]: -32767.000 [Kg/m2] (PDQ:112)
```

5.4 Read L2 High Resolution Product

Precipitation (PRC) product is L2 High Resolution Product.

For Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW), please refer to "5.3 Read L2 Low Resolution Product" on page 51.

5.4.1 C sample program (readL2H_amtk.c)

Sample program (readL2H_amtk.c) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B * Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Geophysical Data for 89A - Geophysical Data for 89B * Pixel Data Quality for 89A - Pixel Data Quality for 89B For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.

For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure "AM2_COMMON_SCANTIME" -> AMTK_getScanTime function
- Output data type is data structure "AM2_COMMON_LATLON" -> AMTK_getLatLon function
- Output data type is float -> AMTK_get_SwathFloat function
- Output data type is unsigned char -> AMTK_get_SwathUchar function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "AMTK.h"
 5
 6 // fixed value
 7 #define LMT 2200 // limit of NumberOfScans
 :
10 int main(int argc, char *argv[]){
11   // interface variable
12   int i,j;      // loop variable
13   int ret;      // return status
14   char buf[512]; // text buffer
15   void *vpnt;   // pointer to void
16   char *fn;     // filename
17   hid_t hnd;    // file handle
18
19   // meta data
20   char geo[512];  // GeophysicalName
21   char gid[512];  // GranuleID
22   char tm1[512];  // ObservationStartDateTime
23   char tm2[512];  // EquatorCrossingDateTime
24   char tm3[512];  // ObservationEndDateTime
25   int num;       // NumberOfScans
26   int ovr;       // OverlapScans

28   // array data
29   AM2_COMMON_SCANTIME st[LMT]; // scantime
30   AM2_COMMON_LATLON ll89a[LMT][AM2_DEF_SNUM_HI];
31   AM2_COMMON_LATLON ll89b[LMT][AM2_DEF_SNUM_HI];
32   float geo1_89a[LMT][AM2_DEF_SNUM_HI];
33   float geo1_89b[LMT][AM2_DEF_SNUM_HI];
34   unsigned char pdq1_89a[LMT][AM2_DEF_SNUM_HI];
35   unsigned char pdq1_89b[LMT][AM2_DEF_SNUM_HI];
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
47   // open
48   hnd=AMTK_openH5(fn);
49   if(hnd<0){
50     printf("AMTK_openH5 error: %s¥n", fn);
51     printf("amtk status: %d¥n", hnd);
52     exit(1);
53   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
55   // read meta: GeophysicalName
56   vpnt=geo;
57   ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
58   if(ret<0){
59     printf("AMTK_getMetaDataName error: GeophysicalName¥n");
60     printf("amtk status: %d¥n",ret);
61     exit(1);
62   }
63   printf("GeophysicalName: %s¥n",geo);
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
111  // read meta: NumberOfScans
112  vpnt=buf;
113  ret=AMTK_getMetaDataName(hnd,"NumberOfScans",(char **)&vpnt);
114  if(ret<0){
115    printf("AMTK_getMetaDataName error: NumberOfScans¥n");
116    printf("amtk status: %d¥n",ret);
117    exit(1);
118  }
119  num=atoi(buf);
120  printf("NumberOfScans: %d¥n",num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

For details to P.14

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
142   // read array: scantime
143   vpnt=st;
144   ret=AMTK_getScanTime(hnd,1,num,(AM2_COMMON_SCANTIME **)&vpnt);
145   if(ret<0){
146     printf("AMTK_getScanTime error.¥n");
147     printf("amtk status: %d¥n",ret);
148     exit(1);
149   }
150   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
151   , st[0].year
152   , st[0].month
153   , st[0].day
154   , st[0].hour
155   , st[0].minute
156   , st[0].second
157   );
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20

Read latitude and longitude.
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_L2_89A (access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
159   // read array: latlon for 89a
160   vpnt=ll89a;
161   ret=AMTK_getLatLon(hnd,(AM2_COMMON_LATLON **)&vpnt,1,num
,AM2_LATLON_L2_89A);
162   if(ret<0){
163     printf("AMTK_getLatLon error: AM2_LATLON_L2_89A¥n");
164     printf("amtk status: %d¥n",ret);
165     exit(1);
166   }
167   printf("latlon89a[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", ll89a[0][0].lat,
ll89a[0][0].lon);
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of geophysical quantities.
Geophysical quantity data is three dimensional (layer * sample * scan) and data from 1 scan to "num" scan is read in the sample program.

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_SWATHA_GEO1 (access label for the first layer of "Geophysical Data for 89A") is shown below.
ret: [Return value] Error: A negative value is returned.

```
179   // read array: geophysical data for 1 layer for 89a
180   vpnt=geo1_89a;
181   ret=AMTK_get_SwathFloat(hnd,(float **)&vpnt,1,num,AM2_SWATHA_GEO1);
182   if(ret<0){
183     printf("AMTK_get_SwathFloat error: AM2_SWATHA_GEO1¥n");
184     printf("amtk status: %d¥n",ret);
185     exit(1);
186   }
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality
Function AMTK_get_SwathUchar and unsigned char type array is used for the acquisition of L2 pixel data quality information.
L2 pixel data quality information is three dimensional (sample * scan * layer), and data from 1 scan to "num" scan is read in the sample program.

ret=AMTK_get_SwathUchar(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Unsigned char type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_A (access label for "Pixel Data Quality for 89A") is shown below.
ret: [Return value] Error: A negative value is returned.

```
197   // read array: pixel data quality for 1 layer for 89a
198   vpnt=pdq1_89a;
199   ret=AMTK_get_SwathUChar(hnd,(unsigned char **)&vpnt,1,num,AM2_PIX_QUAL_A);
200   if(ret<0){
201     printf("AMTK_get_SwathUChar error: AM2_PIX_QUAL_A¥n");
202     printf("amtk status: %d¥n",ret);
203     exit(1);
204   }
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
261   // close
262   ret=AMTK_closeH5(hnd);
```

5.4.2 Compile (Explanation of build_readL2H_amtk_c.sh)

We explain how to compile the C program by using script “build_readL2H_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL2H_amtk.c
16
17 # output filename
18 out=readL2H_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL2H_amtk_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2H_amtk_c.sh
icc -g readL2H_amtk.c -o readL2H_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.4.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2H_amtk_c” is shown as follows.

```
$ ./readL2H_amtk_c GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
GeophysicalName: Precipitation
GranuleID: GW1AM2_201303011809_125D_L2SGPRCHA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans: 1972
limit of NumberOfScans = 2200
OverlapScans: 0
time[scan=0]: 2013/03/01 18:09:10
latlon89a[scan=0][pixel=0]: (  84.4188, -77.9502)
latlon89b[scan=0][pixel=0]: (  84.3305, -78.8925)
geo1_89a[scan=0][pixel=0]: -32767.0 [mm/h] (PDQ: 16)
geo1_89b[scan=0][pixel=0]: -32767.0 [mm/h] (PDQ: 16)
```

5.5 Read L3 Product [Brightness temperature]

5.5.1 C sample program (readL3B_amtk.c)

Sample program (readL3B_amtk.c) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Brightness Temperature (H) - Brightness Temperature (V)

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, you should use AMTK_get_GridFloat function.

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "AMTK.h"
 :
 9 int main(int argc, char *argv[]){
10   // interface variable
11   int i,j;      // loop variable
12   int ret;      // return status
13   char buf[512]; // text buffer
14   void *vpnt;    // pointer to void
15   char *fn;     // filename
16   hid_t hnd;    // file handle
17   int siz[3];   // array size
18   int x;        // grid size x
19   int y;        // grid size y
 :
27   // meta data
28   char geo[512];  // GeophysicalName
29   char gid[512];  // GranuleID
30
31   // array data
32   float *tbH; // brightness temperature for horizontal
33   float *tbV; // brightness temperature for vertical
```

Include header file of AMTK for C language.

Define interface variables for AMTK.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
45   // open
46   hnd=AMTK_openH5(fn);
47   if(hnd<0){
48     printf("AMTK_openH5 error: %s¥n", fn);
49     printf("amtk status: %d¥n", hnd);
50     exit(1);
51   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
53   // read meta: GeophysicalName
54   vpnt=geo;
55   ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
56   if(ret<0){
57     printf("AMTK_getMetaDataName error: GeophysicalName¥n");
58     printf("amtk status: %d¥n",ret);
59     exit(1);
60   }
61   printf("GeophysicalName: %s¥n",geo);
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.
Function AMTK_getDimSize is used for the acquisition of the array size.

ret=AMTK_getDimSize(hnd,label,siz)
hnd: HDF access file id.
label: Access label. AM2_GRID_TBH (access label for “Brightness Temperature (H)”) is shown below.
siz: [Return value] Size of the array.
ret: [Return value] Error: A negative value is returned.

```
86   // get grid size
87   ret=AMTK_getDimSize(hnd,AM2_GRID_TBH,siz);
88   if(ret<0){
89     printf("AMTK_getDimSize error: AM2_GRID_TBH¥n");
90     printf("amtk status: %d¥n",ret);
91     exit(1);
92   }
93   x=siz[1];
94   y=siz[0];
95   printf("grid size x: %d¥n", x);
96   printf("grid size y: %d¥n", y);
```

Allocate the memory.

```
98   // memory allocate
99   tbH=malloc(sizeof(float)*x*y);
100  if(tbH==NULL){
101    printf("memory allocate error: tbH¥n");
102    exit(1);
103  }
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.

ret=AMTK_get_GridFloat(hnd,out,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
label: Access label. AM2_GRID_TBH (access label for “Brightness Temperature (H)”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
110   // read horizontal
111   vpnt=tbH;
112   ret=AMTK_get_GridFloat(hnd,(float **)&vpnt,AM2_GRID_TBH);
113   if(ret<0){
114     printf("AMTK_get_GridFloat error: AM2_GRID_TBH¥n");
115     printf("amtk status: %d¥n",ret);
116     exit(1);
117   }
```

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
212   // memory free
213   free(tbH);
214   free(tbV);
```

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
216   // close
217   ret=AMTK_closeH5(hnd);
```

5.5.2 Compile (Explanation of build_readL3B_amtk_c.sh)

We explain how to compile the C program by using script “build_readL3B_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL3B_amtk.c
16
17 # output filename
18 out=readL3B_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL3B_amtk_c” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3B_amtk_c.sh
icc -g readL3B_amtk.c -o readL3B_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.5.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

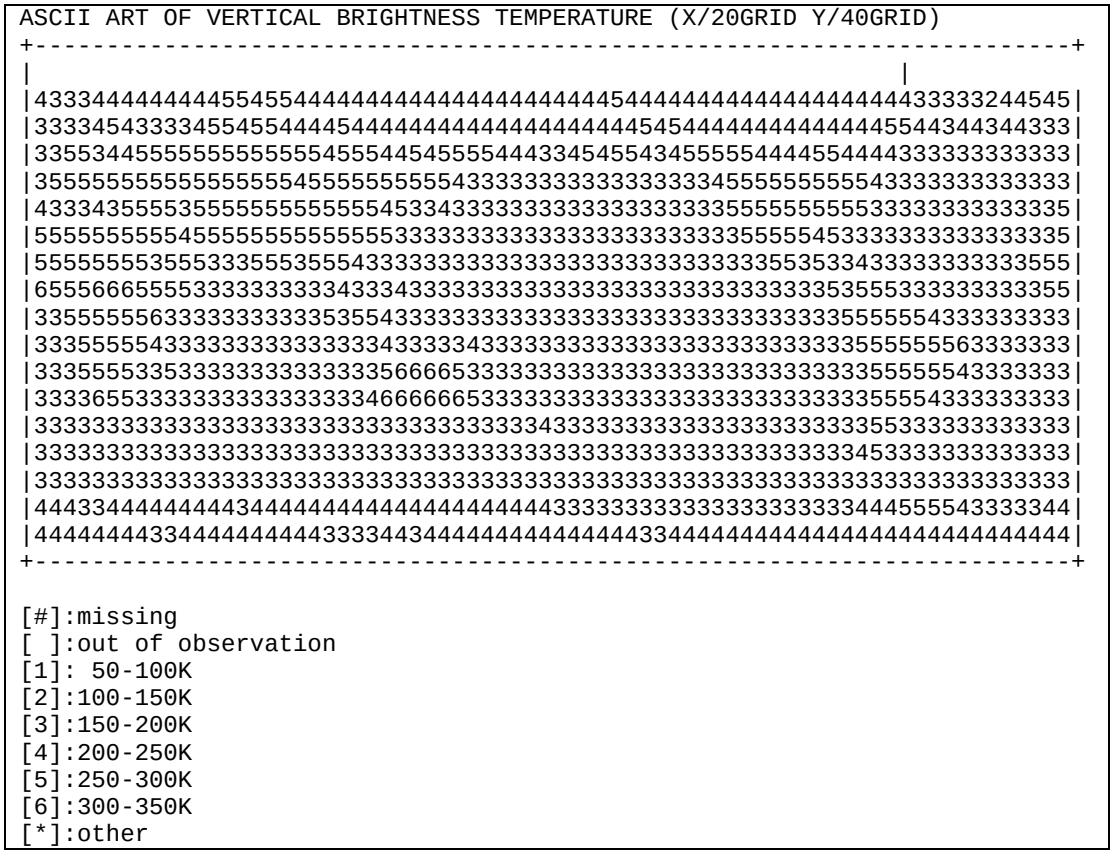
< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3B_amtk_c” is shown as follows.

```
$ ./readL3B_amtk_c GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
GeophysicalName: Brightness Temperature (6GHz)
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110
grid size x: 1440
grid size y: 720

ASCII ART OF HORIZONTAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|411233344443444444444444444444444444444444444444444444433443222222444 4|
|111134311113444444444444444444444444444444444444444444444444434233343111|
|124523445555555444444444444444434312343444144555444444444442112111111111|
|154544444444444444444444444544111111111111111111355444444444222111111112|
|412242554414445444444453411311111111111111111114555444455111111111111115|
|444444444244455544555544111111111111111111111115555535111111111111111124|
|444444424441115551555311111111111111111211111111555151141111111111111444|
|555555555551111211113111211111111111111111111111111111141554111111111155|
|115555555111111111115155311111111111111111111111111111145555541111111111|
|111555553111111111111111421111312111111111111111111111115555555511111111|
|111555521511111111111111455555111111111111111111111111111145555521111111|
|111155511111111111111135555555111111111111111111111111111154552111111111|
|111111111111111111111111111111111131111111111111111111111155111111111111|
|111111111111111111111111111111111111111111111111111111111351111111111111|
|111111111111111111111111111111111111111111111111111111111111111111111111|
|343323444333342434334344444444334332111111111111111111344444441111133|
|333333333333333333333333333333343333333222223333334433443333333333333333|
+------------------------------------------------------------------------+
```


ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)
[#]:missing
[ ]:out of observation
[1]: 50-100K
[2]:100-150K
[3]:150-200K
[4]:200-250K
[5]:250-300K
[6]:300-350K
[*]:other

5.6 Read L3 Product [Geophysical quantity]

5.6.1 C sample program (readL3G_amtk.c)

Sample program (readL3G_amtk.c) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Geophysical Data

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, you should use AMTK_get_GridFloat function.

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "AMTK.h"
 :
 9 int main(int argc, char *argv[]){
10   // interface variable
11   int i,j;      // loop variable
12   int ret;      // return status
13   char buf[512]; // text buffer
14   void *vpnt;    // pointer to void
15   char *fn;     // filename
16   hid_t hnd;    // file handle
17   int siz[3];   // array size
18   int x;        // grid size x
19   int y;        // grid size y
 :
29   // meta data
30   char geo[512];  // GeophysicalName
31   char gid[512];  // GranuleID
32
33   // array data
34   float *geo1; // geophysical data layer 1
35   float *geo2; // geophysical data layer 2
```

Include header file of AMTK for C language.

Define interface variables for AMTK.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
47   // open
48   hnd=AMTK_openH5(fn);
49   if(hnd<0){
50     printf("AMTK_openH5 error: %s¥n", fn);
51     printf("amtk status: %d¥n", hnd);
52     exit(1);
53   }
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

Pointer to pointer variable is passed to functions of AMTK.
First you should modify the pointer, and then pass to a function.

```
55  // read meta: GeophysicalName
56  vpnt=geo;
57  ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
58  if(ret<0){
59    printf("AMTK_getMetaDataName error: GeophysicalName¥n");
60    printf("amtk status: %d¥n",ret);
61    exit(1);
62  }
63  printf("GeophysicalName: %s¥n",geo);
```

To avoid the warnings from the compilers, cast a void pointer to appropriate type.

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.
Function AMTK_getDimSize is used for the acquisition of the array size.

ret=AMTK_getDimSize(hnd,label,siz)
hnd: HDF access file id.
label: Access label. AM2_GRID_GEO1 (access label for the first layer of “Geophysical Data”) is shown below.
siz: [Return value] Size of the array.
ret: [Return value] Error: A negative value is returned.

```
89  // get grid size
90  ret=AMTK_getDimSize(hnd,AM2_GRID_GEO1,siz);
91  if(ret<0){
92    printf("AMTK_getDimSize error: AM2_GRID_GEO1¥n");
93    printf("amtk status: %d¥n",ret);
94    exit(1);
95  }
96  x=siz[1];
97  y=siz[0];
98  printf("grid size x: %d¥n", x);
99  printf("grid size y: %d¥n", y);
```

Allocate the memory.

```
101  // memory allocate layer 1
102  geo1=malloc(sizeof(float)*x*y);
103  if(geo1==NULL){
104    printf("memory allocate error: geo1¥n");
105    exit(1);
106  }
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantity.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

ret=AMTK_get_GridFloat(hnd,out,label)
hnd: HDF access file id.
out: Float type array storing acquired data.
label: Access label. AM2_GRID_GEO1 (access label for the first layer of “Geophysical Data”) and AM2_SWATH_GEO2 (access label for the second layer of “Geophysical Data”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
117   // read layer 1
118   vpnt=geo1;
119   ret=AMTK_get_GridFloat(hnd,(float **)&vpnt,AM2_GRID_GEO1);
120   if(ret<0){
121     printf("AMTK_get_GridFloat error: AM2_GRID_GEO1¥n");
122     printf("amtk status: %d¥n",ret);
123     exit(1);
124   }
125
```

You can read the second layer by changing the access label.

```
126   // read layer 2
127   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
128     vpnt=geo2;
129     ret=AMTK_get_GridFloat(hnd,(float **)&vpnt,AM2_GRID_GEO2);
130     if(ret<0){
131       printf("AMTK_get_GridFloat error: AM2_GRID_GEO2¥n");
132       printf("amtk status: %d¥n",ret);
133       exit(1);
134     }
135   }
```

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
317   // memory free
318   free(geo1);
319   if(strncmp(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
320     free(geo2);
321   }
```

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
323   // close
324   ret=AMTK_closeH5(hnd);
```

5.6.2 Compile (Explanation of build_readL3G_amtk_c.sh)

We explain how to compile the C program by using script “build_readL3G_amtk_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
11 # compiler
12 cc=icc
13
14 # source filename
15 csrc=readL3G_amtk.c
16
17 # output filename
18 out=readL3G_amtk_c
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL3G_amtk_c.sh” is shown in the following. * Line feeds are inserted for convenience.

```
$ ./build_readL3G_amtk_c.sh
icc -g readL3G_amtk.c -o readL3G_amtk_c
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

5.6.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3G_amtk_c” is shown as follows.

```
$ ./readL3G_amtk_c GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000
grid size x: 1440
grid size y: 720

ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/20GRID Y/40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|#21#####################################################################|
|2212###1113#############################################################222|
|1#################################11######3###############011#42223322|
|2#############################0010111241333323#############343413131#|
|#20#######2###############00#12234523442132111 20#########3355421131121#|
|##########################4#32222232321001101111 10#######14*41111201111##|
|########0###000###0####1111111210120111 0*131121100##1#11#01211110100###|
|###########1011111111#211#11111210011121333223224321000#1###1111000000##|
|11#######00010122125#3###1#12232321100010020110001011122#######222321111|
|000######122131112311112#01112#113122123231101000000000110########1001110|
|100####32#12111141111000######223232242121244133111001101 0######11111100|
|1000###121123221125110 0########131112232110121313122011010#####231111001|
|11131212231110111111211111110012102#123122122331121311102 1##011124223242|
|*33222511232222222232233314210224223222110242222222122113##0111421221132|
|10112211222111122211111111122231223223233124332121232322122331223222 2111|
|#######################################3111221222211121122 1########21221##|
|#########################################################################|
+------------------------------------------------------------------------+

[#]:missing
[ ]:out of observation
[0]:  0.000 -  0.030 Kg/m2
[1]:  0.030 -  0.060 Kg/m2
[2]:  0.060 -  0.090 Kg/m2
[3]:  0.090 -  0.120 Kg/m2
[4]:  0.120 -  0.150 Kg/m2
[5]:  0.150 -  0.180 Kg/m2
[*]:other
```

6. FORTRAN90 Programming with AMTK

Letters written in red are explanation of the sample programs

Letters written in blue are explanation of the functions used in the sample programs or basic information of the GCOM-W1 satellite and AMSR2 instrument.

6.1 Read L1B Product

6.1.1 FORTRAN sample program (readL1B_amtk.f)

Sample program (readL1B_amtk.f) which reads the metadata and the datasets of L1B is shown below. Latitude and longitude of low frequency data are also calculated using observation point of 89GHz-A. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans - CoRegistrationParameterA1 - CoRegistrationParameterA2	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B - Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Brightness Temperature (6.9GHz,H) - Brightness Temperature (6.9GHz,V) - Brightness Temperature (7.3GHz,H) - Brightness Temperature (7.3GHz,V) - Brightness Temperature (10.7GHz,H) - Brightness Temperature (10.7GHz,V) - Brightness Temperature (18.7GHz,H) - Brightness Temperature (18.7GHz,V) - Brightness Temperature (23.8GHz,H) - Brightness Temperature (23.8GHz,V) - Brightness Temperature (36.5GHz,H) - Brightness Temperature (36.5GHz,V) - Brightness Temperature (89.0GHz-A,H) - Brightness Temperature (89.0GHz-A,V) - Brightness Temperature (89.0GHz-B,H) - Brightness Temperature (89.0GHz-B,V) * Pixel Data Quality 6 to 36 - Pixel Data Quality 89 * Land_Ocean Flag 6 to 36 - Land_Ocean Flag 89 * Earth Incidence - Earth Azimuth
Data calculated from observation point of 89GHz-A	
- Latitude and longitude (Low mean) - Latitude and longitude (6G) - Latitude and longitude (7G) - Latitude and longitude (10G) - Latitude and longitude (18G) - Latitude and longitude (23G) - Latitude and longitude (36G) For details to P.20	

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure “AM2_COMMON_SCANTIME” -> AMTK_getScanTime function

- Output data type is data structure “AM2_COMMON_LATLON” -> AMTK_getLatLon function
- Output data type is real(4) -> AMTK_get_SwathFloat function
- Output data type is integer(4) -> AMTK_get_SwathInt function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :


 7 C fixed value
 8      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
 9 C interface variable
10      integer(4) i,j        ! loop variable
11      integer(4) ret        ! return status
12      character(len=512) buf ! text buffer
13      character(len=512) fn  ! filename
14      integer(4) hnd        ! file handle
15 C meta data
16      character(len=512) geo  ! GeophysicalName
17      character(len=512) gid  ! GranuleID
18      character(len=512) tm1  ! ObservationStartDateTime
19      character(len=512) tm2  ! EquatorCrossingDateTime
20      character(len=512) tm3  ! ObservationEndDateTime
21      integer(4) num         ! NumberOfScans
22      integer(4) ovr         ! OverlapScans
23      character(len=512) prm1 ! CoRegistrationParameterA1
24      character(len=512) prm2 ! CoRegistrationParameterA2
 :

25 C array data
26      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
27      type(AM2_COMMON_LATLON) ll89a(AM2_DEF_SNUM_HI,LMT) ! latlon for 89a
28      type(AM2_COMMON_LATLON) ll89b(AM2_DEF_SNUM_HI,LMT) ! latlon for 89b
 :
36      real(4) tb06h(AM2_DEF_SNUM_LO,LMT)  ! tb for 06h
37      real(4) tb06v(AM2_DEF_SNUM_LO,LMT)  ! tb for 06v
 :
53      integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT)   ! pixel data quality for 06h
54      integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT)   ! pixel data quality for 06v
 :
63      integer(1) lof06(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 06
64      integer(1) lof07(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 07
 :
72      real(4) ear_in(AM2_DEF_SNUM_LO,LMT) ! earth incidence
73      real(4) ear_az(AM2_DEF_SNUM_LO,LMT) ! earth azimuth
```

Include header file of AMTK for FORTRAN.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
 82 C open
 83       hnd=AMTK_openH5(fn)
 84       if(hnd.lt.0)then
 85         write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
 86         write(*,'(a,i12)')'amtk status: ',hnd
 87         call exit(1)
 88       endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
 89 C read meta: GeophysicalName
 90       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
 91       if(ret.lt.0)then
 92         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
 93         write(*,'(a,i12)')'amtk status: ',ret
 94         call exit(1)
 95       endif
 96       write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
132 C read meta: NumberOfScans
133       ret=AMTK_getMetaDataName(hnd,'NumberOfScans',buf)
134       if(ret.lt.0)then
135         write(*,'(a)')'AMTK_getMetaDataName error: NumberOfScans'
136         write(*,'(a,i12)')'amtk status: ',ret
137         call exit(1)
138       endif
139       read(buf(1:ret),*)num
140       write(*,'(a,i12)')'NumberOfScans: ',num
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Read scanning time. For details to P.14.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
177 C read array: scantime
178       ret=AMTK_getScanTime(hnd,1,num,st)
179       if(ret.lt.0)then
180         write(*,'(a)')'AMTK_getScanTime error.'
181         write(*,'(a,i12)')'amtk status: ',ret
182         call exit(1)
183       endif
184       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
185      +'time(scan=1): '
186      +,st(1)%year
187      +,st(1)%month
188      +,st(1)%day
189      +,st(1)%hour
190      +,st(1)%minute
191      +,st(1)%second
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude. For details to P.20.
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_89A(access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
192 C read array: latlon for 89a
193       ret=AMTK_getLatLon(hnd,ll89a,1,num,AM2_LATLON_89A)
194       if(ret.lt.0)then
195         write(*,'(a)')'AMTK_getLatLon error: AM2_LATLON_89A'
196         write(*,'(a,i12)')'amtk status: ',ret
197         call exit(1)
198       endif
199       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon89a(pixel=1,scan=1): '
200      +,ll89a(1,1)%lat,ll89a(1,1)%lon
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.
Function AMTK_get_SwathFloat and real type array is used for the acquisition of brightness temperature data.
Brightness temperature data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

```
273 C read array: tb for 06h
274       ret=AMTK_get_SwathFloat(hnd,tb06h,1,num,AM2_TB06H)
275       if(ret.lt.0)then
276         write(*,'(a)')'AMTK_get_SwathFloat error: AM2_TB06H'
277         write(*,'(a,i12)')'amtk status: ',ret
278         call exit(1)
279       endif
280       write(*,'(a,f9.2)')'tb06h(pixel=1,scan=1): ',tb06h(1,1)
```

Read L1B pixel data quality * Numbers written on the left are row number of the sample program.

Read L1B pixel data quality
Function AMTK_get_SwathInt and integer type array is used for the acquisition of pixel data quality information.
Pixel data quality information is two dimensional (“Pixel Data Quality 6 to 36”: (sample * 16) * scan, “Pixel Data Quality 89”: (sample * 8) * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathInt(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Integer type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_LO (access label for “Pixel Data Quality 6 to 36”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
401 C read array: pixel data quality for low
402       ret=AMTK_get_SwathInt(hnd,pdqlo,1,num,AM2_PIX_QUAL_LO)
403       if(ret.lt.0)then
404         write(*,'(a)')'AMTK_get_SwathInt error: AM2_PIX_QUAL_LO'
405         write(*,'(a,i12)')'amtk status: ',ret
406         call exit(1)
407       endif
```

“Pixel Data Quality 6 to 36” are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to integer(1) type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
408       do j=1,num
409         do i=1,AM2_DEF_SNUM_LO
410           pdq06v(i,j)=pdqlo((i-1)*16+ 2,j)*10+pdqlo((i-1)*16+ 1,j)
411           pdq06h(i,j)=pdqlo((i-1)*16+ 4,j)*10+pdqlo((i-1)*16+ 3,j)
412           pdq07v(i,j)=pdqlo((i-1)*16+ 6,j)*10+pdqlo((i-1)*16+ 5,j)
413           pdq07h(i,j)=pdqlo((i-1)*16+ 8,j)*10+pdqlo((i-1)*16+ 7,j)
414         enddo
415       enddo
```

Read L1B land ocean flag * Numbers written on the left are row number of the sample program.

Read L1B land ocean flag.
Function AMTK_get_SwathInt and integer type array is used for the acquisition of land ocean flag data.
Land ocean flag data is two dimensional ("Land/Ocean Flag 6 to 36": sample * (scan * 6), "Land/Ocean Flag 89": sample * (scan * 2)), and data from 1 scan to "num" scan is read in the sample program.

For details to P.18.

```
439 C read array: land ocean flag for low
440       ret=AMTK_get_SwathInt(hnd,loflo,1,num,AM2_LOF_LO)
441       if(ret.lt.0)then
442         write(*,'(a)')'AMTK_get_SwathInt error: AM2_LOF_LO'
443         write(*,'(a,i12)')'amtk status: ',ret
444         call exit(1)
445       endif
```

"Land/Ocean Flag 6 to 36" and "Land/Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
446       do j=1,num
447         do i=1,AM2_DEF_SNUM_LO
448           lof06(i,j)=loflo(i,num*0+j)
449           lof07(i,j)=loflo(i,num*1+j)
450           lof10(i,j)=loflo(i,num*2+j)
451           lof18(i,j)=loflo(i,num*3+j)
452           lof23(i,j)=loflo(i,num*4+j)
453           lof36(i,j)=loflo(i,num*5+j)
454         enddo
455       enddo
```

Read earth incidence * Numbers written on the left are row number of the sample program.

Read earth incidence.
Function AMTK_get_SwathFloat and float type array is used for the acquisition of earth incidence data.
Earth incidence data is two dimensional (sample * scan), and data from 1 scan to "num" scan is read in the sample program.

```
477 C read array: earth incidence
478       ret=AMTK_get_SwathFloat(hnd,ear_in,1,num,AM2_EARTH_INC)
479       if(ret.lt.0)then
480         write(*,'(a)')'AMTK_get_SwathFloat error: AM2_EARTH_INC'
481         write(*,'(a,i12)')'amtk status: ',ret
482         call exit(1)
483       endif
484       write(*,'(a,f9.2)')'ear_in(pixel=1,scan=1): ',ear_in(1,1)
```

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
493 C close
494       ret=AMTK_closeH5(hnd)
```

6.1.2 Compile (Explanation of build_readL1B_amtk_f.sh)

We explain how to compile the Fortran program by using script “build_readL1B_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) is required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL1B_amtk.f
16
17 # output filename
18 out=readL1B_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL1B_amtk_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1B_amtk_f.sh
ifort -g readL1B_amtk.f -o readL1B_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.1.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1B_amtk_f” is shown as follows.

```
$ ./readL1B_amtk_f GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGBTBR_000000
ObservationStartDateTime: 2012-07-26T11:45:43.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:37.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans:        1979
limit of NumberOfScans =   2200
OverlapScans:          20
CoRegistrationParameterA1: 6G-1.25000,7G-1.00000,10G-1.25000,18G-1.25000,23G-1.25000,36G-1.00000
CoRegistrationParameterA2: 6G-0.00000,7G--0.10000,10G--0.25000,18G-0.00000,23G--0.25000,36G-0.00000
time(scan=1): 2012/07/26 11:45:43
latlon89a(pixel=1,scan=1): ( -73.3289, 136.7714)
latlon89b(pixel=1,scan=1): ( -73.4038, 137.1498)
latlonlm(pixel=1,scan=1): ( -73.3538, 136.6228)
latlon06(pixel=1,scan=1): ( -73.3592, 136.6213)
latlon07(pixel=1,scan=1): ( -73.3497, 136.6429)
latlon10(pixel=1,scan=1): ( -73.3506, 136.6001)
latlon18(pixel=1,scan=1): ( -73.3592, 136.6213)
latlon23(pixel=1,scan=1): ( -73.3506, 136.6001)
latlon36(pixel=1,scan=1): ( -73.3532, 136.6514)
tb06h(pixel=1,scan=1):    173.28
tb06v(pixel=1,scan=1):    208.22
tb07h(pixel=1,scan=1):    173.07
tb07v(pixel=1,scan=1):    207.54
tb10h(pixel=1,scan=1):    170.94
tb10v(pixel=1,scan=1):    204.95
tb18h(pixel=1,scan=1):    164.85
tb18v(pixel=1,scan=1):    199.84
tb23h(pixel=1,scan=1):    163.22
tb23v(pixel=1,scan=1):    196.53
tb36h(pixel=1,scan=1):    156.56
tb36v(pixel=1,scan=1):    186.39
tb89ah(pixel=1,scan=1):    163.76
tb89av(pixel=1,scan=1):    179.27
tb89bh(pixel=1,scan=1):    170.60
tb89bv(pixel=1,scan=1):    188.16
```

```
pdq06h(pixel=1,scan=1):     0
pdq06v(pixel=1,scan=1):     0
pdq07h(pixel=1,scan=1):     0
pdq07v(pixel=1,scan=1):     0
pdq10h(pixel=1,scan=1):     0
pdq10v(pixel=1,scan=1):     0
pdq18h(pixel=1,scan=1):     0
pdq18v(pixel=1,scan=1):     0
pdq23h(pixel=1,scan=1):     0
pdq23v(pixel=1,scan=1):     0
pdq36h(pixel=1,scan=1):     0
pdq36v(pixel=1,scan=1):     0
pdq89ah(pixel=1,scan=1):     0
pdq89av(pixel=1,scan=1):     0
pdq89bh(pixel=1,scan=1):     0
pdq89bv(pixel=1,scan=1):     0
lof06(pixel=1,scan=1):  100
lof07(pixel=1,scan=1):  100
lof10(pixel=1,scan=1):  100
lof18(pixel=1,scan=1):  100
lof23(pixel=1,scan=1):  100
lof36(pixel=1,scan=1):  100
lof89a(pixel=1,scan=1):  100
lof89b(pixel=1,scan=1):  100
ear_in(pixel=1,scan=1):        55.20
ear_az(pixel=1,scan=1):       144.76
```

6.2 Read L1R product

6.2.1 FORTRAN sample program (readL1R_amtk.f)

Sample program (readL1R_amtk.f) which reads the metadata and the datasets of L1R is shown below. Latitude and longitude of low frequency data are extracted from observation point of 89GHz-A. Values of data are dumped to the standard output.

Only part of L1R brightness temperature data are handled in this sample program. Please refer to "3.8 Level1 Resampling Products (L1R)" on page 18.

<table>
<tr><th>Metadata</th><th>Datasets</th></tr>
<tr><td>* GeophysicalName
- GranuleID
- ObservationStartDateTime
- EquatorCrossingDateTime
- ObservationEndDateTime
* NumberOfScans
- OverlapScans
- CoRegistrationParameterA1
- CoRegistrationParameterA2</td><td rowspan="3">* Scan Time
* Latitude of Observation Point for 89A
- Latitude of Observation Point for 89B
- Longitude of Observation Point for 89A
- Longitude of Observation Point for 89B
* Brightness Temperature (res06,6.9GHz,H)
- Brightness Temperature (res06,6.9GHz,V)
- Brightness Temperature (res06,7.3GHz,H)
- Brightness Temperature (res06,7.3GHz,V)
- Brightness Temperature (res10,10.7GHz,H)
- Brightness Temperature (res10,10.7GHz,V)
- Brightness Temperature (res23,18.7GHz,H)
- Brightness Temperature (res23,18.7GHz,V)
- Brightness Temperature (res23,23.8GHz,H)
- Brightness Temperature (res23,23.8GHz,V)
- Brightness Temperature (res36,36.5GHz,H)
- Brightness Temperature (res36,36.5GHz,V)
- Brightness Temperature (res36,89.0GHz,H)
- Brightness Temperature (res36,89.0GHz,V)
- Brightness Temperature (original,89GHz-A,H)
- Brightness Temperature (original,89GHz-A,V)
- Brightness Temperature (original,89GHz-B,H)
- Brightness Temperature (original,89GHz-B,V)
* Pixel Data Quality 6 to 36
- Pixel Data Quality 89
* Land_Ocean Flag 6 to 36
- Land_Ocean Flag 89
* Earth Incidence
- Earth Azimuth</td></tr>
<tr><td>Data extracted from observation point of 89GHz-A</td></tr>
<tr><td>- Latitude and longitude of low frequency data
For details to P.20.</td></tr>
</table>

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure "AM2_COMMON_SCANTIME" -> AMTK_getScanTime function
- Output data type is data structure "AM2_COMMON_LATLON" -> AMTK_getLatLon function
- Output data type is real(4) -> AMTK_get_SwathFloat function

- Output data type is integer(4) -> AMTK_get_SwathInt function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :

 7 C fixed value
 8      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
 9 C interface variable
10      integer(4) i,j        ! loop variable
11      integer(4) ret        ! return status
12      character(len=512) buf ! text buffer
13      character(len=512) fn  ! filename
14      integer(4) hnd        ! file handle
15 C meta data
16      character(len=512) geo  ! GeophysicalName
17      character(len=512) gid  ! GranuleID
18      character(len=512) tm1  ! ObservationStartDateTime
19      character(len=512) tm2  ! EquatorCrossingDateTime
20      character(len=512) tm3  ! ObservationEndDateTime
21      integer(4) num         ! NumberOfScans
22      integer(4) ovr         ! OverlapScans
23      character(len=512) prm1 ! CoRegistrationParameterA1
24      character(len=512) prm2 ! CoRegistrationParameterA2

25 C array data
26      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
27      type(AM2_COMMON_LATLON) ll89ar(AM2_DEF_SNUM_HI,LMT) ! latlon for 89a
28      type(AM2_COMMON_LATLON) ll89br(AM2_DEF_SNUM_HI,LMT) ! latlon for 89b
 :
30      real(4) tb06h06(AM2_DEF_SNUM_LO,LMT) ! tb for 06h, resolution 06G
31      real(4) tb06v06(AM2_DEF_SNUM_LO,LMT) ! tb for 06v, resolution 06G
 :
49      integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT)   ! pixel data quality for 06h
50      integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT)   ! pixel data quality for 06v
 :
59      integer(1) lof06(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 06
60      integer(1) lof10(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 10
 :
66      real(4) ear_in(AM2_DEF_SNUM_LO,LMT) ! earth incidence
67      real(4) ear_az(AM2_DEF_SNUM_LO,LMT) ! earth azimuth
```

Include header file of AMTK for FORTRAN.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
 76 C open
 77       hnd=AMTK_openH5(fn)
 78       if(hnd.lt.0)then
 79         write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
 80         write(*,'(a,i12)')'amtk status: ',hnd
 81         call exit(1)
 82       endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
 83 C read meta: GeophysicalName
 84       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
 85       if(ret.lt.0)then
 86         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
 87         write(*,'(a,i12)')'amtk status: ',ret
 88         call exit(1)
 89       endif
 90       write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Read number of scans from metadata.
Number of scan is necessary when reading datasets.

```
126 C read meta: NumberOfScans
127       ret=AMTK_getMetaDataName(hnd,'NumberOfScans',buf)
128       if(ret.lt.0)then
129         write(*,'(a)')'AMTK_getMetaDataName error: NumberOfScans'
130         write(*,'(a,i12)')'amtk status: ',ret
131         call exit(1)
132       endif
133       read(buf(1:ret),*)num
134       write(*,'(a,i12)')'NumberOfScans: ',num
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

For details to P.14.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
171 C read array: scantime
172       ret=AMTK_getScanTime(hnd,1,num,st)
173       if(ret.lt.0)then
174         write(*,'(a)')'AMTK_getScanTime error.'
175         write(*,'(a,i12)')'amtk status: ',ret
176         call exit(1)
177       endif
178       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
179      +'time(scan=1): '
180      +,st(1)%year
181      +,st(1)%month
182      +,st(1)%day
183      +,st(1)%hour
184      +,st(1)%minute
185      +,st(1)%second
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20.

Read latitude and longitude.
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_89A(access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
186 C read array: latlon for 89a altitude revised
187       ret=AMTK_getLatLon(hnd,ll89ar,1,num,AM2_LATLON_RS_89A)
188       if(ret.lt.0)then
189         write(*,'(a)')'AMTK_getLatLon error: AM2_LATLON_RS_89A'
190         write(*,'(a,i12)')'amtk status: ',ret
191         call exit(1)
192       endif
193       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon89ar(pixel=1,scan=1): '
194      +,ll89ar(1,1)%lat,ll89ar(1,1)%lon
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.
Function AMTK_get_SwathFloat and real type array is used for the acquisition of brightness temperature data.
Brightness temperature data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

```
213 C read array: tb for 06h, resolution 06G
214       ret=AMTK_get_SwathFloat(hnd,tb06h06,1,num,AM2_RES06_TB06H)
215       if(ret.lt.0)then
216         write(*,'(a)')'AMTK_get_SwathFloat error: AM2_RES06_TB06H'
217         write(*,'(a,i12)')'amtk status: ',ret
218         call exit(1)
219       endif
220       write(*,'(a,f9.2)')'tb06h06(pixel=1,scan=1): ',tb06h06(1,1)
```

Read L1R pixel data quality * Numbers written on the left are row number of the sample program.

Read L1R pixel data quality
Function AMTK_get_SwathInt and integer type array is used for the acquisition of pixel data quality information.
Pixel data quality information is two dimensional (“Pixel Data Quality 6 to 36”: (sample * 16) * scan, “Pixel Data Quality 89”: (sample * 8) * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathInt(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Integer type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_LO (access label for “Pixel Data Quality 6 to 36”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
357 C read array: pixel data quality for low
358       ret=AMTK_get_SwathInt(hnd,pdqlo,1,num,AM2_PIX_QUAL_LO)
359       if(ret.lt.0)then
360         write(*,'(a)')'AMTK_get_SwathInt error: AM2_PIX_QUAL_LO'
361         write(*,'(a,i12)')'amtk status: ',ret
362         call exit(1)
363       endif
```

“Pixel Data Quality 6 to 36” are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to integer(1) type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
364       do j=1,num
365         do i=1,AM2_DEF_SNUM_LO
366           pdq06v(i,j)=pdqlo((i-1)*16+ 2,j)*10+pdqlo((i-1)*16+ 1,j)
367           pdq06h(i,j)=pdqlo((i-1)*16+ 4,j)*10+pdqlo((i-1)*16+ 3,j)
368           pdq07v(i,j)=pdqlo((i-1)*16+ 6,j)*10+pdqlo((i-1)*16+ 5,j)
369           pdq07h(i,j)=pdqlo((i-1)*16+ 8,j)*10+pdqlo((i-1)*16+ 7,j)
370         enddo
371       enddo
```

Read L1R land ocean flag * Numbers written on the left are row number of the sample program.

Read L1R land ocean flag. For details to P.18.
Function AMTK_get_SwathInt and integer type array is used for the acquisition of land ocean flag data.
Land ocean flag data is two dimensional ("Land/Ocean Flag 6 to 36": sample * (scan * 4), "Land/Ocean Flag 89": sample * (scan * 2)), and data from 1 scan to "num" scan is read in the sample program.

```
395 C read array: land ocean flag for low
396       ret=AMTK_get_SwathInt(hnd,loflo,1,num,AM2_LOF_RES_LO)
397       if(ret.lt.0)then
398         write(*,'(a)')'AMTK_get_SwathInt error: AM2_LOF_RES_LO'
399         write(*,'(a,i12)')'amtk status: ',ret
400         call exit(1)
401       endif
```

"Land/Ocean Flag 6 to 36" and "Land/Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
402       do j=1,num
403         do i=1,AM2_DEF_SNUM_LO
404           lof06(i,j)=loflo(i,num*0+j)
405           lof10(i,j)=loflo(i,num*1+j)
406           lof23(i,j)=loflo(i,num*2+j)
407           lof36(i,j)=loflo(i,num*3+j)
408         enddo
409       enddo
```

Read earth incidence * Numbers written on the left are row number of the sample program.

Read earth incidence.
Function AMTK_get_SwathFloat and real type array is used for the acquisition of earth incidence data.
Earth incidence data is two dimensional (sample * scan), and data from 1 scan to "num" scan is read in the sample program.

```
429 C read array: earth incidence
430       ret=AMTK_get_SwathFloat(hnd,ear_in,1,num,AM2_EARTH_INC)
431       if(ret.lt.0)then
432         write(*,'(a)')'AMTK_get_SwathFloat error: AM2_EARTH_INC'
433         write(*,'(a,i12)')'amtk status: ',ret
434         call exit(1)
435       endif
436       write(*,'(a,f9.2)')'ear_in(pixel=1,scan=1): ',ear_in(1,1)
```

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
445 C close
446       ret=AMTK_closeH5(hnd)
```

6.2.2 Compile (Explanation of build_readL1R_amtk_f.sh)

We explain how to compile the Fortran program by using script “build_readL1R_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) are required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL1R_amtk.f
16
17 # output filename
18 out=readL1R_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL1R_amtk_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1R_amtk_f.sh
ifort -g readL1R_amtk.f -o readL1R_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.2.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1R_amtk_f” is shown as follows.

```
$ ./readL1R_amtk_f GW1AM2_201207261145_055A_
L1SGRTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGRTB
R_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGRTBR
_0000000
ObservationStartDateTime: 2012-07-26T11:45:4
3.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:3
7.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.
735Z
NumberOfScans:        1979
limit of NumberOfScans =   2200
OverlapScans:          20
CoRegistrationParameterA1: 6G-0.00000,7G-0.0
0000,10G-0.00000,18G-0.00000,23G-0.00000,36G
-0.00000
CoRegistrationParameterA2: 6G-0.00000,7G-0.0
0000,10G-0.00000,18G-0.00000,23G-0.00000,36G
-0.00000
time(scan=1): 2012/07/26 11:45:43
latlon89ar(pixel=1,scan=1): ( -73.3581, 136.8
432)
latlon89br(pixel=1,scan=1): ( -73.4328, 137.2
216)
latlonlr(pixel=1,scan=1): ( -73.3581, 136.843
2)
tb06h06(pixel=1,scan=1):    173.41
tb06v06(pixel=1,scan=1):    208.09
tb07h06(pixel=1,scan=1):    173.07
tb07v06(pixel=1,scan=1):    207.11
tb10h10(pixel=1,scan=1):    170.40
tb10v10(pixel=1,scan=1):    204.58
tb18h23(pixel=1,scan=1):    165.83
tb18v23(pixel=1,scan=1):    199.41
tb23h23(pixel=1,scan=1):    163.55
tb23v23(pixel=1,scan=1):    195.90
tb36h36(pixel=1,scan=1):    153.55
tb36v36(pixel=1,scan=1):    183.95
tb89h36(pixel=1,scan=1):    163.86
tb89v36(pixel=1,scan=1):    181.05
tb89ah(pixel=1,scan=1):    163.76
tb89av(pixel=1,scan=1):    179.27
tb89bh(pixel=1,scan=1):    170.60
tb89bv(pixel=1,scan=1):    188.16
pdq06h(pixel=1,scan=1):   0
pdq06v(pixel=1,scan=1):   0
pdq07h(pixel=1,scan=1):   0
pdq07v(pixel=1,scan=1):   0
pdq10h(pixel=1,scan=1):   0
pdq10v(pixel=1,scan=1):   0
pdq18h(pixel=1,scan=1):   0
pdq18v(pixel=1,scan=1):   0
pdq23h(pixel=1,scan=1):   0
pdq23v(pixel=1,scan=1):   0
pdq36h(pixel=1,scan=1):   0
pdq36v(pixel=1,scan=1):   0
pdq89ah(pixel=1,scan=1):   0
pdq89av(pixel=1,scan=1):   0
pdq89bh(pixel=1,scan=1):   0
pdq89bv(pixel=1,scan=1):   0
lof06(pixel=1,scan=1): 100
lof10(pixel=1,scan=1): 100
lof23(pixel=1,scan=1): 100
lof36(pixel=1,scan=1): 100
lof89a(pixel=1,scan=1): 100
lof89b(pixel=1,scan=1): 100
ear_in(pixel=1,scan=1):    55.20
ear_az(pixel=1,scan=1):   144.76
```

6.3 Read L2 Low Resolution Product

L2 Low Resolution Products are Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW). For Precipitation(PRC) product, please refer to “6.4 Read L2 High Resolution Product” on page 110.

6.3.1 Fortran sample program (readL2L_amtk.f)

Sample program (readL2L_amtk.f)で which reads the metadata and the datasets of L1B is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point * Longitude of Observation Point * Geophysical Data * Pixel Data Quality For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure “AM2_COMMON_SCANTIME” -> AMTK_getScanTime function
- Output data type is data structure “AM2_COMMON_LATLON” -> AMTK_getLatLon function
- Output data type is real(4) -> AMTK_get_SwathFloat function
- Output data type is integer(1) -> AMTK_get_SwathUchar function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :


 7 C fixed value
 8      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
 :
12 C interface variable
13      integer(4) i,j        ! loop variable
14      integer(4) ret        ! return status
15      character(len=512) buf ! text buffer
16      character(len=512) fn  ! filename
17      integer(4) hnd        ! file handle
18 C meta data
19      character(len=512) geo  ! GeophysicalName
20      character(len=512) gid  ! GranuleID
21      character(len=512) tm1  ! ObservationStartDateTime
22      character(len=512) tm2  ! EquatorCrossingDateTime
23      character(len=512) tm3  ! ObservationEndDateTime
24      integer(4) num         ! NumberOfScans
25      integer(4) ovr         ! OverlapScans

26 C array data
27      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
28      type(AM2_COMMON_LATLON) ll(AM2_DEF_SNUM_LO,LMT) ! latlon

29      real(4) geo1(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 1
30      real(4) geo2(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 2

31      integer(4) pdq1(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 1
32      integer(4) pdq2(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 2
33      integer(1) pdqtmp(AM2_DEF_SNUM_LO,LMT*2) ! pixel data quality temporary
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
42 C open
43       hnd=AMTK_openH5(fn)
44       if(hnd.lt.0)then
45         write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
46         write(*,'(a,i12)')'amtk status: ',hnd
47         call exit(1)
48       endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
49 C read meta: GeophysicalName
50       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
51       if(ret.lt.0)then
52         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
53         write(*,'(a,i12)')'amtk status: ',ret
54         call exit(1)
55       endif
56       write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
103 C read meta: NumberOfScans
104       ret=AMTK_getMetaDataName(hnd,'NumberOfScans',buf)
105       if(ret.lt.0)then
106         write(*,'(a)')'AMTK_getMetaDataName error: NumberOfScans'
107         write(*,'(a,i12)')'amtk status: ',ret
108         call exit(1)
109       endif
110       read(buf(1:ret),*)num
111       write(*,'(a,i12)')'NumberOfScans: ',num
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

For details to P.14.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
128 C read array: scantime
129       ret=AMTK_getScanTime(hnd,1,num,st)
130       if(ret.lt.0)then
131         write(*,'(a)')'AMTK_getScanTime error.'
132         write(*,'(a,i12)')'amtk status: ',ret
133         call exit(1)
134       endif
135       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
136      +'time(scan=1): '
137      +,st(1)%year
138      +,st(1)%month
139      +,st(1)%day
140      +,st(1)%hour
141      +,st(1)%minute
142      +,st(1)%second
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20.

Read latitude and longitude
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_L2_LO (access label for “Latitude of Observation Point” and “Longitude of Observation Point”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
143 C read array: latlon
144       ret=AMTK_getLatLon(hnd,ll,1,num,AM2_LATLON_L2_LO)
145       if(ret.lt.0)then
146         write(*,'(a)')'AMTK_getLatLon error: AM2_LATLON_L2_LO'
147         write(*,'(a,i12)')'amtk status: ',ret
148         call exit(1)
149       endif
150       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon(pixel=1,scan=1): '
151      +,ll(1,1)%lat,ll(1,1)%lon
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Function AMTK_get_SwathFloat and real type array is used for the acquisition of geophysical quantity data.
Geophysical quantity data is three dimensional (layer * sample * scan) and data from 1 scan to "num" scan is read in the sample program.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is "Snow Water Equivalent".
Geophysical quantity stored in the second layer of sea surface temperature product is "SST obtained by 10GHz".

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Real type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_SWATH_GEO1 (access label for the first layer of "Geophysical Data") and AM2_SWATH_GEO2 (access label for the second layer of "Geophysical Data") is shown below.
ret: [Return value] Error: A negative value is returned.

```
152 C read array: geophysical data for 1 layer
153       if((gid(30:32).ne.'SND').and.(gid(30:32).ne.'SST'))then
154         ret=AMTK_get_SwathFloat(hnd,geo1,1,num,AM2_SWATH_GEO1)
155         if(ret.lt.0)then
156           write(*,'(a)')'AMTK_get_SwathFloat error: AM2_SWATH_GEO1'
157           write(*,'(a,i12)')'amtk status: ',ret
158           call exit(1)
159         endif
160       endif
```

You can read the second layer by changing the access label.

```
161 C read array: geophysical data for 2 layer
162       if((gid(30:32).eq.'SND').or.(gid(30:32).eq.'SST'))then
163         ! layer 1
164         ret=AMTK_get_SwathFloat(hnd,geo1,1,num,AM2_SWATH_GEO1)
165         if(ret.lt.0)then
166           write(*,'(a)')'AMTK_get_SwathFloat error: AM2_SWATH_GEO1'
167           write(*,'(a,i12)')'amtk status: ',ret
168           call exit(1)
169         endif
170         ! layer 2
171         ret=AMTK_get_SwathFloat(hnd,geo2,1,num,AM2_SWATH_GEO2)
172         if(ret.lt.0)then
173           write(*,'(a)')'AMTK_get_SwathFloat error: AM2_SWATH_GEO2'
174           write(*,'(a,i12)')'amtk status: ',ret
175           call exit(1)
176         endif
177       endif
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.
Function AMTK_get_SwathUchar and integer(1) type array is used for the acquisition of L2 pixel data quality information.
L2 pixel data quality information is three dimensional (sample * scan * layer) and data from 1 scan to “num” scan is read in the sample program.
Snow depth product and Sea surface temperature product have two layer of pixel data quality.

ret=AMTK_get_SwathUchar(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Integer type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL (access label for “Pixel Data Quality”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
178 C read array: pixel data quality for 1 layer
179       if((gid(30:32).ne.'SND').and.(gid(30:32).ne.'SST'))then
180         ! read
181         ret=AMTK_get_SwathUChar(hnd,pdqtmp,1,num,AM2_PIX_QUAL)
182         if(ret.lt.0)then
183           write(*,'(a)')'AMTK_get_SwathUChar error: AM2_PIX_QUAL'
184           write(*,'(a,i12)')'amtk status: ',ret
185           call exit(1)
186         endif
187         ! convert signed to unsigned
188         do j=1,num
189           do i=1,AM2_DEF_SNUM_LO
190             pdq1(i,j)=pdqtmp(i,j)
191             if(pdq1(i,j).ge.0)then
192               pdq1(i,j)=pdq1(i,j)
193             else
194               pdq1(i,j)=pdq1(i,j)+256
195             endif
196           enddo
197         enddo
198       endif
```

The data value of L2 pixel data quality (0 to 255) is read as (-128 to 127) , because unsigned integer types is not supported by Fortran. Use the following data manipulation to extract the actual value.

When data value is between 0 to 127, keep its default value.

When data value is between -128 to -1, add 256 to the data value.

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
199 C read array: pixel data quality for 2 layer
200       if((gid(30:32).eq.'SND').or.(gid(30:32).eq.'SST'))then
201         ! read
202         ret=AMTK_get_SwathUChar(hnd,pdqtmp,1,num,AM2_PIX_QUAL)
203         if(ret.lt.0)then
204           write(*,'(a)')'AMTK_get_SwathUChar error: AM2_PIX_QUAL'
205           write(*,'(a,i12)')'amtk status: ',ret
206           call exit(1)
207         endif
208         ! separate & convert signed to unsigned
209         do j=1,num
210           do i=1,AM2_DEF_SNUM_LO
211             pdq1(i,j)=pdqtmp(i,num*0+j)
212             pdq2(i,j)=pdqtmp(i,num*1+j)
213             if(pdq1(i,j).ge.0)then
214               pdq1(i,j)=pdq1(i,j)
215             else
216               pdq1(i,j)=pdq1(i,j)+256
217             endif
218             if(pdq2(i,j).ge.0)then
219               pdq2(i,j)=pdq2(i,j)
220             else
221               pdq2(i,j)=pdq2(i,j)+256
222             endif
223           enddo
224         enddo
225       endif
```

Close HDF file

* Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
257 C close
258       ret=AMTK_closeH5(hnd)
```

6.3.2 Compile (Explanation of build_readL2L_amtk_f.sh)

We explain how to compile the Fortran program by using script “build_readL2L_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) is required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL2L_amtk.f
16
17 # output filename
18 out=readL2L_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL2L_amtk_f.sh” is shown in the following. * Line feeds are inserted for convenience.

```
$ ./build_readL2L_amtk_f.sh
ifort -g readL2L_amtk.f -o readL2L_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.3.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2L_amtk_f” is shown as follows.

```
$ ./readL2L_amtk_f GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_201303011809_125D_L2SGCLWLA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans:        1972
limit of NumberOfScans =   2200
OverlapScans:           0
time(scan=1): 2013/03/01 18:09:10
latlon(pixel=1,scan=1): (  84.4574, -78.1076)
geo1(pixel=1,scan=1): -32767.000 [Kg/m2] (PDQ:112)
```

6.4 Read L2 High Resolution Product

Precipitation (PRC) product is L2 High Resolution Product.

For Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW), please refer to "6.3 Read L2 Low Resolution Product" on page 101.

6.4.1 Fortran sample program (readL2H_amtk.f)

Sample program (readL2H_amtk.f) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B * Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Geophysical Data for 89A - Geophysical Data for 89B * Pixel Data Quality for 89A - Pixel Data Quality for 89B For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, it is necessary to use suitable function shown below.

- Output data type is data structure "AM2_COMMON_SCANTIME" -> AMTK_getScanTime function
- Output data type is data structure "AM2_COMMON_LATLON" -> AMTK_getLatLon function
- Output data type is real(4) -> AMTK_get_SwathFloat function
- Output data type is integer(1) -> AMTK_get_SwathUchar function

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :


 7 C fixed value
 8      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
 :
11 C interface variable
12      integer(4) i,j        ! loop variable
13      integer(4) ret        ! return status
14      character(len=512) buf ! text buffer
15      character(len=512) fn  ! filename
16      integer(4) hnd        ! file handle
17 C meta data
18      character(len=512) geo  ! GeophysicalName
19      character(len=512) gid  ! GranuleID
20      character(len=512) tm1  ! ObservationStartDateTime
21      character(len=512) tm2  ! EquatorCrossingDateTime
22      character(len=512) tm3  ! ObservationEndDateTime
23      integer(4) num         ! NumberOfScans
24      integer(4) ovr         ! OverlapScans

25 C array data
26      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
27      type(AM2_COMMON_LATLON) ll89a(AM2_DEF_SNUM_HI,LMT)
28      type(AM2_COMMON_LATLON) ll89b(AM2_DEF_SNUM_HI,LMT)
29      real(4) geo1_89a(AM2_DEF_SNUM_HI,LMT)
30      real(4) geo1_89b(AM2_DEF_SNUM_HI,LMT)
31      integer(4) pdq1_89a(AM2_DEF_SNUM_HI,LMT)
32      integer(4) pdq1_89b(AM2_DEF_SNUM_HI,LMT)
33      integer(1) pdqtmp(AM2_DEF_SNUM_HI,LMT)
```

Include header file of AMTK for C language.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for AMTK.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Use “AM2_COMMON_LATLON” data structure for the acquisition of latitude and longitude data.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in AMTK, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in AMTK, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
42 C open
43       hnd=AMTK_openH5(fn)
44       if(hnd.lt.0)then
45         write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
46         write(*,'(a,i12)')'amtk status: ',hnd
47         call exit(1)
48       endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
49 C read meta: GeophysicalName
50       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
51       if(ret.lt.0)then
52         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
53         write(*,'(a,i12)')'amtk status: ',ret
54         call exit(1)
55       endif
56       write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
97 C read meta: NumberOfScans
98       ret=AMTK_getMetaDataName(hnd,'NumberOfScans',buf)
99       if(ret.lt.0)then
100        write(*,'(a)')'AMTK_getMetaDataName error: NumberOfScans'
101        write(*,'(a,i12)')'amtk status: ',ret
102        call exit(1)
103      endif
104      read(buf(1:ret),*)num
105      write(*,'(a,i12)')'NumberOfScans: ',num
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Read scanning time.
Function AMTK_getScanTime and data structure of AM2_COMMON_SCANTIME is used for the acquisition of scanning time data.
Scanning time data is one dimensional, and data from 1 scan to “num” scan is read in the sample program.
For details to P.14

ret=AMTK_getScanTime(hnd,bgn,end,out)
hnd: HDF access file id.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
out: Data structure storing scanning time data.
ret: [Return value] Error: A negative value is returned.

```
122 C read array: scantime
123       ret=AMTK_getScanTime(hnd,1,num,st)
124       if(ret.lt.0)then
125         write(*,'(a)')'AMTK_getScanTime error.'
126         write(*,'(a,i12)')'amtk status: ',ret
127         call exit(1)
128       endif
129       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
130      +'time(scan=1): '
131      +,st(1)%year
132      +,st(1)%month
133      +,st(1)%day
134      +,st(1)%hour
135      +,st(1)%minute
136      +,st(1)%second
```

Read latitude and longitude * Numbers written on the left are row number of the sample program.

For details to P.20
Read latitude and longitude
Function AMTK_getLatLon and Data structure of AM2_COMMON_LATLON is used for the acquisition of latitude and longitude data.
Latitude and longitude data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_getLatLon(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Data structure storing latitude and longitude data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_LATLON_L2_89A (access label for “Latitude of Observation Point for 89A” and “Longitude of Observation Point for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
137 C read array: latlon for 89a
138       ret=AMTK_getLatLon(hnd,ll89a,1,num,AM2_LATLON_L2_89A)
139       if(ret.lt.0)then
140         write(*,'(a)')'AMTK_getLatLon error: AM2_LATLON_L2_89A'
141         write(*,'(a,i12)')'amtk status: ',ret
142         call exit(1)
143       endif
144       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon89a(pixel=1,scan=1): '
145      +,ll89a(1,1)%lat,ll89a(1,1)%lon
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Function AMTK_get_SwathFloat and real type array is used for the acquisition of geophysical quantities.
Brightness temperature data is three dimensional (layer * sample * scan) and data from 1 scan to "num" scan is read in the sample program.

ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Real type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_SWATHA_GEO1 (access label for the first layer of "Geophysical Data for 89A") is shown below.
ret: [Return value] Error: A negative value is returned.

```
155 C read array: geophysical data for 1 layer for 89a
156       ret=AMTK_get_SwathFloat(hnd,geo1_89a,1,num,AM2_SWATHA_GEO1)
157       if(ret.lt.0)then
158         write(*,'(a)')'AMTK_get_SwathFloat error: AM2_SWATHA_GEO1'
159         write(*,'(a,i12)')'amtk status: ',ret
160         call exit(1)
161       endif
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality
Function AMTK_get_SwathUchar and integer(1) type array is used for the acquisition of L2 pixel data quality information.
L2 pixel data quality information is three dimensional (sample * scan * layer) and data from 1 scan to “num” scan is read in the sample program.

ret=AMTK_get_SwathUchar(hnd,out,bgn,end,label)
hnd: HDF access file id.
out: Integer(1) type array storing acquired data.
bgn: Scan number of acquisition start.
end: Scan number of acquisition end.
label: Access label. AM2_PIX_QUAL_A (access label for “Pixel Data Quality for 89A”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
169 C read array: pixel data quality for 1 layer for 89a
170       ! read
171       ret=AMTK_get_SwathUChar(hnd,pdqtmp,1,num,AM2_PIX_QUAL_A)
172       if(ret.lt.0)then
173         write(*,'(a)')'AMTK_get_SwathUChar error: AM2_PIX_QUAL_A'
174         write(*,'(a,i12)')'amtk status: ',ret
175         call exit(1)
176       endif
177       ! convert signed to unsigned
178       do j=1,num
179         do i=1,AM2_DEF_SNUM_HI
180           pdq1_89a(i,j)=pdqtmp(i,j)
181           if(pdq1_89a(i,j).ge.0)then
182             pdq1_89a(i,j)=pdq1_89a(i,j)
183           else
184             pdq1_89a(i,j)=pdq1_89a(i,j)+256
185           endif
186         enddo
187       enddo
```

The data value of L2 pixel data quality (0 to 255) is read as (-128 to 127) , because unsigned integer types is not supported by Fortran. Use the following data manipulation to extract the actual value.

When data value is between 0 to 127, keep its default value.

When data value is between -128 to -1, add 256 to the data value.

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
257 C close
258       ret=AMTK_closeH5(hnd)
```

6.4.2 Compile (Explanation of build_readL2H_amtk_f.sh)

We explain how to compile the C program by using script “build_readL2H_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) is required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL2H_amtk.f
16
17 # output filename
18 out=readL2H_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL2H_amtk_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2H_amtk_f.sh
ifort -g readL2H_amtk.f -o readL2H_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.4.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2H_amtk_f” is shown as follows.

```
$ ./readL2H_amtk_f GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
GeophysicalName: Precipitation
GranuleID: GW1AM2_201303011809_125D_L2SGPRCHA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans:        1972
limit of NumberOfScans =   2200
OverlapScans:           0
time(scan=1): 2013/03/01 18:09:10
latlon89a(pixel=1,scan=1): (  84.4188, -77.9502)
latlon89b(pixel=1,scan=1): (  84.3305, -78.8925)
geo1_89a(pixel=1,scan=1): -32767.0 [mm/h] (PDQ: 16)
geo1_89b(pixel=1,scan=1): -32767.0 [mm/h] (PDQ: 16)
```

6.5 Read L3 Product [Brightness temperature]

6.5.1 Fortran sample program (readL3B_amtk.f)

Sample program (readL3B_amtk.f) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Brightness Temperature (H) - Brightness Temperature (V)

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, you should use AMTK_get_GridFloat function.

AMTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :
10 C interface variable
11      integer(4) i,j        ! loop variable
12      integer(4) ret        ! return status
13      character(len=512) buf ! text buffer
14      character(len=512) fn  ! filename
15      integer(4) hnd        ! file handle
16      integer(4) siz(3)     ! array size
17      integer(4) x          ! grid size x
18      integer(4) y          ! grid size y
 :
24 C meta data
25      character(len=512) geo  ! GeophysicalName
26      character(len=512) gid  ! GranuleID

27 C array data
28      real(4),allocatable::tbH(:,:)
29      real(4),allocatable::tbV(:,:)
```

Include header file of AMTK for FORTRAN.

Define interface variables for AMTK.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
38 C open
39      hnd=AMTK_openH5(fn)
40      if(hnd.lt.0)then
41        write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
42        write(*,'(a,i12)')'amtk status: ',hnd
43        call exit(1)
44      endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
45 C read meta: GeophysicalName
46       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
47       if(ret.lt.0)then
48         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
49         write(*,'(a,i12)')'amtk status: ',ret
50         call exit(1)
51       endif
52     write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.
Function AMTK_getDimSize is used for the acquisition of the array size.

ret=AMTK_getDimSize(hnd,label,siz)
hnd: HDF access file id.
label: Access label. AM2_GRID_TBH (access label for “Brightness Temperature (H)”) is shown below.
siz: [Return value] Size of the array.
ret: [Return value] Error: A negative value is returned.

```
72 C get grid size
73       ret=AMTK_getDimSize(hnd,AM2_GRID_TBH,siz);
74       if(ret.lt.0)then
75         write(*,'(a)')'AMTK_getDimSize error: AM2_GRID_TBH'
76         write(*,'(a,i12)')'amtk status: ',ret
77         call exit(1)
78       endif
79       x=siz(2);
80       y=siz(1);
81       write(*,'(a,i12)')'grid size x: ',x
82       write(*,'(a,i12)')'grid size y: ',y
```

Allocate the memory.

```
83 C memory allocate
84       allocate(tbH(x,y),stat=ret)
85       if(ret.ne.0)then
86         write(*,'(a)')'memory allocate error: tbH'
87         call exit(1)
88       endif
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Read brightness temperature.

ret=AMTK_get_GridFloat(hnd,out,label)
hnd: HDF access file id.
out: Real type array storing acquired data.
label: Access label. AM2_GRID_TBH (access label for “Brightness Temperature (H)”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
 94 C read horizontal
 95      ret=AMTK_get_GridFloat(hnd,tbH,AM2_GRID_TBH)
 96      if(ret.lt.0)then
 97        write(*,'(a)')'AMTK_get_GridFloat error: AM2_GRID_TBH'
 98        write(*,'(a,i12)')'amtk status: ',ret
 99        call exit(1)
100      endif
```

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
200 C memory free
201      deallocate(tbH)
202      deallocate(tbV)
```

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
203 C close
204      ret=AMTK_closeH5(hnd)
```

6.5.2 Compile (Explanation of build_readL3B_amtk_f.sh)

We explain how to compile the Fortran program by using script “build_readL3B_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) is required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL3B_amtk.f
16
17 # output filename
18 out=readL3B_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL3B_amtk_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3B_amtk_f.sh
ifort -g readL3B_amtk.f -o readL3B_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.5.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

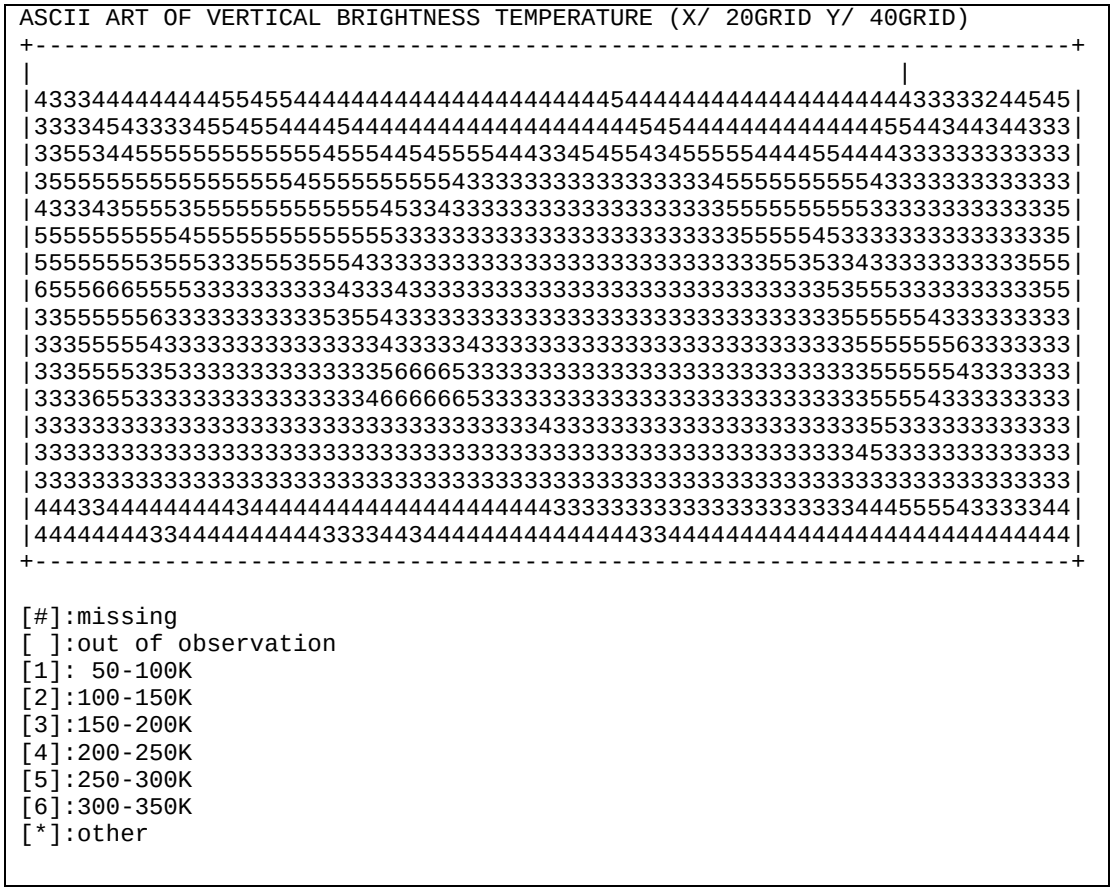
< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3B_amtk_f” is shown as follows.

```
$ ./readL3B_amtk_f GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
GeophysicalName: Brightness Temperature (6GHz)
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110
grid size x:       1440
grid size y:        720

ASCII ART OF HORIZONTAL BRIGHTNESS TEMPERATURE (X/ 20GRID Y/ 40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|411233344443444444444444444444444444444444444444444444433443222222444 4|
|111134311113444444444444444444444444444444444444444444444444434233343111|
|124523445555555444444444444444434312343444144555444444444442112111111111|
|154544444444444444444444444544111111111111111111355444444444222111111112|
|412242554414445444444453411311111111111111111114555444455111111111111115|
|444444444244455544555544111111111111111111111115555535111111111111111124|
|444444424441115551555311111111111111111121111111155151141111111111111444|
|555555555551111211113111211111111111111111111111111111141554111111111155|
|115555555111111111115155311111111111111111111111111111145555541111111111|
|111555553111111111111111421111312111111111111111111111155555555111111111|
|111555521511111111111111455555111111111111111111111111145555521111111111|
|111155511111111111111135555555111111111111111111111111154552111111111111|
|111111111111111111111111111111111131111111111111111111155111111111111111|
|111111111111111111111111111111111111111111111111111111351111111111111111|
|111111111111111111111111111111111111111111111111111111111111111111111111|
|343323444333342434334344444444334332111111111111111111344444441111133|
|333333333333333333333333333333343333333222223333334433443333333333333333|
+------------------------------------------------------------------------+
```


ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/ 20GRID Y/ 40GRID)
[#]:missing
[ ]:out of observation
[1]: 50-100K
[2]:100-150K
[3]:150-200K
[4]:200-250K
[5]:250-300K
[6]:300-350K
[*]:other

6.6 Read L3 Product [Geophysical quantity]

6.6.1 Fortran sample program (readL3G_amtk.f)

Sample program (readL3G_amtk.f) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Geophysical Data

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

To acquire the metadata, you should use AMTK_getMetaDataName function.
For the acquisition of datasets, you should use AMTK_get_GridFloat function.

MTK reads the datasets with specifying the HDF access label. Access label is an identifier to access HDF dataset which is defined in AMTK.

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      implicit none
 3 C include
 4      include 'AMTK_f.h'
 :
10 C interface variable
11      integer(4) i,j        ! loop variable
12      integer(4) ret        ! return status
13      character(len=512) buf ! text buffer
14      character(len=512) fn  ! filename
15      integer(4) hnd        ! file handle
16      integer(4) siz(3)     ! array size
17      integer(4) x          ! grid size x
18      integer(4) y          ! grid size y
 :
26 C meta data
27      character(len=512) geo  ! GeophysicalName
28      character(len=512) gid  ! GranuleID

29 C array data
30      real(4),allocatable::geo1(:,:)
31      real(4),allocatable::geo2(:,:)
```

Include header file of AMTK for FORTRAN.

Define interface variables for AMTK.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fn: AMSR2 HDF file name.
hnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
40 C open
41      hnd=AMTK_openH5(fn)
42      if(hnd.lt.0)then
43        write(*,'(a,a)')'AMTK_openH5 error: ',fn(1:len_trim(fn))
44        write(*,'(a,i12)')'amtk status: ',hnd
45        call exit(1)
46      endif
```

Read metadata * Numbers written on the left are row number of the sample program.

Read the metadata.

ret=AMTK_getMetaDataName(hnd,met,out)
hnd: HDF access file id.
met: Metadata name.
out: Metadata value.
ret: [Return value] Normal: The number of meta data character is returned. Error: A negative value is returned.

```
47 C read meta: GeophysicalName
48       ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
49       if(ret.lt.0)then
50         write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
51         write(*,'(a,i12)')'amtk status: ',ret
52         call exit(1)
53       endif
54     write(*,'(a,a)')'GeophysicalName: ',geo(1:len_trim(geo))
```

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.
Function AMTK_getDimSize is used for the acquisition of the array size.

ret=AMTK_getDimSize(hnd,label,siz)
hnd: HDF access file id.
label: Access label. AM2_GRID_GEO1 (access label for the first layer of “Geophysical Data”) is shown below.
siz: [Return value] Size of the array.
ret: [Return value] Error: A negative value is returned.

```
75 C get grid size
76       ret=AMTK_getDimSize(hnd,AM2_GRID_GEO1,siz);
77       if(ret.lt.0)then
78         write(*,'(a)')'AMTK_getDimSize error: AM2_GRID_GEO1'
79         write(*,'(a,i12)')'amtk status: ',ret
80         call exit(1)
81       endif
82       x=siz(2);
83       y=siz(1);
84       write(*,'(a,i12)')'grid size x: ',x
85       write(*,'(a,i12)')'grid size y: ',y
```

Allocate the memory.

```
86 C memory allocate layer 1
87       allocate(geo1(x,y),stat=ret)
88       if(ret.ne.0)then
89         write(*,'(a)')'memory allocate error: geo1'
90         call exit(1)
91       endif
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantity.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

ret=AMTK_get_GridFloat(hnd,out,label)
hnd: HDF access file id.
out: Real type array storing acquired data.
label: Access label. AM2_GRID_GEO1 (access label for the first layer of “Geophysical Data”) and AM2_SWATH_GEO2 (access label for the second layer of “Geophysical Data”) is shown below.
ret: [Return value] Error: A negative value is returned.

```
100 C read layer 1
101       ret=AMTK_get_GridFloat(hnd,geo1,AM2_GRID_GEO1)
102       if(ret.lt.0)then
103         write(*,'(a)')'AMTK_get_GridFloat error: AM2_GRID_GEO1'
104         write(*,'(a,i12)')'amtk status: ',ret
105         call exit(1)
106       endif
```

You can read the second layer by changing the access label.

```
107 C read layer 2
108       if(gid(30:32).eq.'SND')then
109         ret=AMTK_get_GridFloat(hnd,geo2,AM2_GRID_GEO2)
110         if(ret.lt.0)then
111           write(*,'(a)')'AMTK_get_GridFloat error: AM2_GRID_GEO2'
112           write(*,'(a,i12)')'amtk status: ',ret
113           call exit(1)
114         endif
115       endif
```

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
301 C memory free
302       deallocate(geo1)
303       if(gid(30:32).eq.'SND')then
304         deallocate(geo2)
305       endif
```

Close the HDF5 file.

ret=AMTK_closeH5(hnd)
hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

```
306 C close
307       ret=AMTK_closeH5(hnd)
```

6.6.2 Compile (Explanation of build_readL3G_amtk_f.sh)

We explain how to compile the Fortran program by using script “build_readL3G_amtk_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-9.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
 8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 9 SZIP=/home/user1/util/szip_2.1
10
```

Specify the compiler you use in row number 12.
Intel compiler (ifort) or PGI compiler (pgf90) is required.

```
11 # compiler
12 fc=ifort
13
14 # source filename
15 fsrc=readL3G_amtk.f
16
17 # output filename
18 out=readL3G_amtk_f
19
20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"
22
23 # compile
24 cmd="$fc -g $fsrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-L$AMTK/lib -L$HDF5/lib -L$SZIP/lib $lib"
25 echo $cmd
26 $cmd
27
28 # garbage
29 rm -f *.o
```

The execution example of “build_readL3G_amtk_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3G_amtk_f.sh
ifort -g readL3G_amtk.f -o readL3G_amtk_f
-I/home/user1/util/AMTK_AMSR2_1.11/include
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/AMTK_AMSR2_1.11/lib
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

6.6.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3G_amtk_f” is shown as follows.

```
$ ./readL3G_amtk_f GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000
grid size x:       1440
grid size y:        720

ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/ 20GRID Y/ 40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|#21#####################################################################|
|2212###1113#############################################################222|
|1#################################11######3###############011#42223322|
|2#############################0010111241333323#############343413131#|
|#20#######2###############00#12234523442132111 20#########3355421131121#|
|##########################4#32222232321001101111107#######14*41111201111##|
|########0###000###0####1111111210120111 0*131121100##1#11#01211110100###|
|###########1011111111#211#11111210011121333223224321000#1###1111000000##|
|11#######00010122125#3###1#12232321100010020110001011122#######222321111|
|000######122131112311112#01112#113122123231101000000000110########1001110|
|100####32#12111141111000######2232322421212441331110011010######11111100|
|1000###12112322112511 00########131112232110121313122011010#####231111001|
|11131212231110111111211111110012102#123122122331121311102 1##011124223242|
|*33222511232222222232233314210224223222110242222222122113##0111421221132|
|10112211222111122211111111122231223223233124332121232322122331223222 2111|
|#######################################3111221222211121122 1########21221##|
|#########################################################################|
+------------------------------------------------------------------------+

[#]:missing
[ ]:out of observation
[0]:  0.000 -  0.030 Kg/m2
[1]:  0.030 -  0.060 Kg/m2
[2]:  0.060 -  0.090 Kg/m2
[3]:  0.090 -  0.120 Kg/m2
[4]:  0.120 -  0.150 Kg/m2
[5]:  0.150 -  0.180 Kg/m2
[*]:other
```

7. C Programming with HDF5 library (without AMTK)

Letters written in red are explanation of the sample programs

Letters written in blue are explanation of the functions used in the sample programs or basic information of the GCOM-W1 satellite and AMSR2 instrument.

7.1 Read L1B Product

7.1.1 C sample program (readL1B_hdf5.c)

Sample program (readL1B_hdf5.c) which read the metadata and the datasets of L1B is shown below. Latitude and longitude of low frequency data are also calculated using observation point of 89GHz-A. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93 and calculation of low frequency latitude and longitude.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans - CoRegistrationParameterA1 - CoRegistrationParameterA2	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B - Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Brightness Temperature (6.9GHz,H) - Brightness Temperature (6.9GHz,V) - Brightness Temperature (7.3GHz,H) - Brightness Temperature (7.3GHz,V) - Brightness Temperature (10.7GHz,H) - Brightness Temperature (10.7GHz,V) - Brightness Temperature (18.7GHz,H) - Brightness Temperature (18.7GHz,V) - Brightness Temperature (23.8GHz,H) - Brightness Temperature (23.8GHz,V) - Brightness Temperature (36.5GHz,H) - Brightness Temperature (36.5GHz,V) - Brightness Temperature (89.0GHz-A,H) - Brightness Temperature (89.0GHz-A,V) - Brightness Temperature (89.0GHz-B,H) - Brightness Temperature (89.0GHz-B,V) * Pixel Data Quality 6 to 36 - Pixel Data Quality 89 * Land_Ocean Flag 6 to 36 - Land_Ocean Flag 89 * Earth Incidence - Earth Azimuth
Data calculated from observation point of 89GHz-A	
* Latitude and longitude (Low mean) - Latitude and longitude (6G) - Latitude and longitude (7G) - Latitude and longitude (10G) - Latitude and longitude (18G) - Latitude and longitude (23G) - Latitude and longitude (36G) → For details to P.20	

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "hdf5.h"
 4 #include "amsr2time.h"
 5 #include "amsr2latlon.h"
 6
 7 // fixed value
 8 #define LMT 2200 // limit of NumberOfScans
 9 #define AM2_DEF_SNUM_HI 486 // high resolution pixel width
10 #define AM2_DEF_SNUM_LO 243 // low  resolution pixel width
11
12 int main(int argc, char *argv[]){
13   // interface variable
14   int i,j;         // loop variable
15   herr_t ret;      // return status
16   char *buf = NULL; // text buffer
17   char *fn  = NULL; // filename
18   hid_t fhnd;      // file handle
19   hid_t ahnd;      // attribute handle
20   hid_t atyp;      // attribute type
21   hid_t dhnd;      // dataset handle
22   hid_t shnd;      // dataspace handle
23
24   // meta data
25   char *geo = NULL; // GeophysicalName
26   char *gid = NULL; // GranuleID
27   char *tm1 = NULL; // ObservationStartDateTime
28   char *tm2 = NULL; // EquatorCrossingDateTim
29   char *tm3 = NULL; // ObservationEndDateTime
30   int num;         // NumberOfScans
31   int ovr;         // OverlapScans
32   double prm1[7];   // CoRegistrationParameterA1
33   double prm2[7];   // CoRegistrationParameterA2

40   // array data
41   AM2_COMMON_SCANTIME st[LMT]; // scantime
42   float lat89a[LMT][AM2_DEF_SNUM_HI]; // lat for 89a
43   float lat89b[LMT][AM2_DEF_SNUM_HI]; // lat for 89b

51   float lon89a[LMT][AM2_DEF_SNUM_HI]; // lon for 89a
52   float lon89b[LMT][AM2_DEF_SNUM_HI]; // lon for 89b

60   float tb06h [LMT][AM2_DEF_SNUM_LO]; // tb for 06h
61   float tb06v [LMT][AM2_DEF_SNUM_LO]; // tb for 06v

76   unsigned char pdq06h [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06h
77   unsigned char pdq06v [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06v

85   unsigned char lof06 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 06
86   unsigned char lof07 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 07

94   float ear_in[LMT][AM2_DEF_SNUM_LO]; // earth incidence
95   float ear_az[LMT][AM2_DEF_SNUM_LO]; // earth azimuth
```

Include header file of HDF5 for C language.

Include header files of subroutines for time conversion and calculating the position of low frequency data.

Limit of scan is sufficient in number 2200. When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
107   // hdf5 initialize
108   ret = H5open();
109   if(ret < 0){
110     printf("h5open error: %d¥n" ,ret);
111     exit(1);
112   }
113
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allows read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
114   // open
115   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
116   if(fhnd < 0){
117     printf("H5Fopen error: %s¥n", fn);
118     exit(1);
119   }
```

Read metadata * Numbers written on the left are row number of the sample program.

First you have to open the attributes which are stored with the data object using H5Aopen function to acquire metadata.
Acquire the attribute type next, and you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
121   // read meta: GeophysicalName
122   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
123   atyp = H5Aget_type(ahnd);
124   ret = H5Aread(ahnd, atyp, &geo);
125   if(ret < 0){
126     printf("H5Aread error: GeophysicalName¥n");
127     exit(1);
128   }
129   ret = H5Aclose(ahnd);
130   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
176   // read meta: NumberOfScans
177   ahnd = H5Aopen(fhnd, "NumberOfScans", H5P_DEFAULT);
178   atyp = H5Aget_type(ahnd);
179   ret = H5Aread(ahnd, atyp, &buf);
180   if(ret < 0){
181     printf("H5Aread error: NumberOfScans¥n");
182     exit(1);
183   }
184   ret = H5Aclose(ahnd);
185   num = atoi(buf);
186   printf("NumberOfScans: %d¥n", num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read Co-registration parameter from metadata.
These are necessary when calculating latitude and longitude of low frequency data.

```
209   // read meta: CoRegistrationParameterA1
210   ahnd = H5Aopen(fhnd, "CoRegistrationParameterA1", H5P_DEFAULT);
211   atyp = H5Aget_type(ahnd);
212   ret = H5Aread(ahnd, atyp, &buf);
213   if(ret < 0){
214     printf("H5Aread error: CoRegistrationParameterA1¥n");
215     exit(1);
216   }
217   ret = H5Aclose(ahnd);
```

For details to P.20.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
Overlap scans contained at both end of L1 data should be removed.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

For details to P.13.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array. Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

```
265   // read array: scantime
26    dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
26    ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
26    if(ret < 0){
26      printf("H5Dread error: Scan
27      exit(1);
27    }
27    ret = H5Dclose(dhnd);
27    // cutoff overlap
27    for(j = 0; j < num; ++j){
27      r8d1[j] = r8d1[j + ovr];
27    }
27    for(j = num; j < LMT; ++j){
27      r8d1[j] = 0;
27    }
28    // convert
28    amsr2time_(&num, r8d1, st);
28    // sample display
28    printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
284   , st[0].year
285   , st[0].month
286   , st[0].day
287   , st[0].hour
288   , st[0].minute
289   , st[0].second
290   );
```

Read scanning time.

Remove overlap scans.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude of 89GHz.
Overlap scans contained at both end of L1 data should be removed.
Latitudes and longitudes of low frequency data are calculated from position of 89GHz-A and the co-registration parameters.

```
292   // read array: latlon for 89a
293   // read lat
294   dhnd = H5Dopen(fhnd ,"Latitude of Observation Point for 89A", H5P_DEFAULT);
295   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat89a);
296   if(ret < 0){
297     printf("H5Dread error: Latitude of Observation Point for 89A¥n");
298     exit(1);
299   }
300   ret = H5Dclose(dhnd);
301   // read lon
302   dhnd = H5Dopen(fhnd ,"Longitude of Observation Point for 89A", H5P_DEFAULT);
303   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon89a);
304   if(ret < 0){
305     printf("H5Dread error: Longitude of Observation Point for 89A¥n");
306     exit(1);
307   }
308   ret = H5Dclose(dhnd);
309   // cutoff overlap
310   for(j = 0; j < num; ++j){
311     for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
312       lat89a[j][i] = lat89a[j+ovr][i];
313       lon89a[j][i] = lon89a[j+ovr][i];
314     }
315   }
316   for(j = num; j < LMT; ++j){
317     for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
318       lat89a[j][i] = 0;
319       lon89a[j][i] = 0;
320     }
321   }
  :
```

Read latitude.

Read longitude.

Remove overlap scans.

AMSR2 has frequency channels of 6, 7, 10, 18, 23, 36, and 89GHz and their observation point are not strictly same.
Latitude and longitude stored in L1 products are that of 89GHz. Position of low frequency data can be calculated from position of 89GHz-A and the co-registration parameters.
Subroutine for calculating the position of low frequency data is prepared in sample programs. This calculation is automatically applied when using AMTK. For details to P.20.

```
358   read array: latlon for low mean
359   sr2latlon_(&num, &prm1[6], &prm2[6], lat89a, lon89a, latlm, lonlm);
360   intf("latlonlm[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", latlm[0][0],
lonlm[0][0]);
```

Calculate the position of low frequency data.

Calculating the position of low frequency data.
amsr2latlon_(num,prm1,prm2,lat89a,lon89a,latlow,lonlow)
num: Number of scan.
prm1: CoRegistrationParameterA1 for each frequency(6G/7G/10G/18G/23G/36G/Low mean)
prm2: CoRegistrationParameterA2 for each frequency(6G/7G/10G/18G/23G/36G/Low mean)
lat89a: Latitude of 89GHz-A.
lon89a: Longitude of 89GHz-A.
latlow: [Return value] Latitude of specified frequency.
lonlow: [Return value] Longitude of specified frequency.

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer (0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value(65534) must be excluded.

```
386   // read array: tb for 06h
387   hnd = H5Dopen(fhnd, "Brightness Temperature (6.9GHz,H)", H5P_DEFAULT);
388   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
389   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
390   e(ahnd);
391   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, tb06h);
392   if(ret < 0){
393    printf("H5Dread error: Brightness Temperature (6.9GHz,H)¥n");
394    exit(1);
395   }
396   ret = H5Dclose(dhnd);
397   // cutoff overlap & change scale
398   for(j = 0; j < num; ++j){
399    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
400     tb06h[j][i] = tb06h[j+ovr][i];
401     if(tb06h[j][i] < 65534) tb06h[j][i] = tb06h[j][i] * sca;
402    }
403   }
404   for(j = num; j < LMT; ++j){
405    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
406     tb06h[j][i] = 0;
407    }
408   }
409   // sample display
410   printf("tb06h[scan=0][pixel=0]: %9.2f¥n", tb06h[0][0]);
```

Read scale factor.

Read brightness temperature.

Remove overlap scans.

Exclude the missing value (65535) and error value(65534) when scale handling is applied.

Scale handling is automatically applied when using AMTK. For details to P.22.

Read L1B pixel data quality * Numbers written on the left are row number of the sample program.

"Pixel Data Quality 6 to 36" consist of two 1-byte integer and 12 out of 16 bits shows bit by bit each frequency and polarization.
"Pixel Data Quality 89" consist of one 1-byte integer and 4 out of 8 bits shows bit by bit each frequency and polarization.

```
802   // read array: pixel data quality for low
803   dhnd = H5Dopen(fhnd, "Pixel Data Quality 6 to 36", H5P_DEFAULT);
804   ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, i1d2hi);
805   if(ret < 0){
806     printf("H5Dread error: Pixel Data Quality 6 to 36¥n");
807     exit(1);
808   }
809   ret = H5Dclose(dhnd);
```

Read L1B pixel data quality

```
810   // cutoff overlap & separate
811   for(j = 0; j < num; ++j){
812     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
813       pdq06v[j][i]=0;
814       if((i1d2hi[j+ovr][i*2+0] &   1) != 0) pdq06v[j][i]=1;
815       if((i1d2hi[j+ovr][i*2+0] &   2) != 0) pdq06v[j][i]=10;
816       if(((i1d2hi[j+ovr][i*2+0] &   1) != 0) && ((i1d2hi[j+ovr][i*2+0] &   2) !=
0)) pdq06v[j][i]=11;
817       pdq06h[j][i]=0;
818       if((i1d2hi[j+ovr][i*2+0] &   4) != 0) pdq06h[j][i]=1;
819       if((i1d2hi[j+ovr][i*2+0] &   8) != 0) pdq06h[j][i]=10;
820       if(((i1d2hi[j+ovr][i*2+0] &   4) != 0) && ((i1d2hi[j+ovr][i*2+0] &   8) !=
0)) pdq06h[j][i]=11;
821       pdq07v[j][i]=0;
822       if((i1d2hi[j+ovr][i*2+0] &  16) != 0) pdq07v[j][i]=1;
823       if((i1d2hi[j+ovr][i
824       if(((i1d2hi[j+ovr][                                    [i*2+0] &  32) !=
0)) pdq07v[j][i]=11;
825       pdq07h[j][i]=0;
826       if((i1d2hi[j+ovr][i*2+0] &  64) != 0) pdq07h[j][i]=1;
827       if((i1d2hi[j+ovr][i*2+0] & 128) != 0) pdq07h[j][i]=10;
828       if(((i1d2hi[j+ovr][i*2+0] &  64) != 0) && ((i1d2hi[j+ovr][i*2+0] & 128) !=
0)) pdq07h[j][i]=11;
829     }
830   }
831   for(j = num; j < LMT; ++j){
832     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
833       pdq06h[j][i]=0;
834       pdq06v[j][i]=0;
835       pdq07h[j][i]=0;
836       pdq07v[j][i]=0;
837     }
838   }
```

Split the bit by bit information to each array of frequency and polarization.

Remove overlap scans also applied.

Read L1B land ocean flag * Numbers written on the left are row number of the sample program.

Data type of L1B land ocean flag is 1-byte integer and two dimensional (sample * (scan * channel)).
"Land_Ocean Flag 6 to 36" has 6 frequency channels (6, 7, 10, 18, 23, and 36GHz).
"Land_Ocean Flag 89" has 2 frequency channels (89GHz-A and 89GHz-B).

For details to P.18.

```
876   // read array: land ocean flag for low
87    dhnd = H5Dopen(fhnd, "Land_Ocean Flag 6 to 36", H5P_DEFAULT);
878   ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, loflo);
879   if(ret < 0){
880     printf("H5Dread error: Land_Ocean Flag 6 to 36¥n");
881     exit(1);
882   }
883   ret = H5Dclose(dhnd);
```

Read L1B land ocean flag.

"Land_Ocean Flag 6 to 36" and "Land_Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
884   // separate
885   for(j = 0; j < num+ovr*2; ++j){
886     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
887       lof06[j][i]=loflo[(num+ovr*2)*0+j][i];
888       lof07[j][i]=loflo[(num+ovr*2)*1+j][i];
889       lof10[j][i]=loflo[(num+ovr*2)*2+j][i];
890       lof18[j][i]=loflo[(num+ovr*2)*3+j][i];
891       lof23[j][i]=loflo[(num+ovr*2)*4+j][i];
892       lof36[j][i]=loflo[(num+ovr*2)*5+j][i];
893     }
894   }
895   // cutoff overlap
896   for(j = 0; j < num; ++j){
897     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
898       lof06[j][i]=lof06[j+ovr][i];
899       lof07[j][i]=lof07[j+ovr][i];
900       lof10[j][i]=lof10[j+ovr][i];
901       lof18[j][i]=lof18[j+ovr][i];
902       lof23[j][i]=lof23[j+ovr][i];
903       lof36[j][i]=lof36[j+ovr][i];
904     }
905   }
906   for(j = num; j < LMT; ++j){
907     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
908       lof06[j][i]=0;
909       lof07[j][i]=0;
91        lof10[j][i]=0;
91        lof18[j][i]=0;
912       lof23[j][i]=0;
913       lof36[j][i]=0;
914     }
915   }
```

Split the data to each frequency.

Remove overlap scans also applied.

Read earth incidence * Numbers written on the left are row number of the sample program.

Earth incidence is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the earth incidence data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
 956   // read array: earth incidence
 957   dhnd = H5Dopen(fhnd, "Earth Incidence", H5P_DEFAULT);
 958   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT); // get scale
 959   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);       // (
 960   ret = H5Aclose(ahnd);                               //
 961   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
ear_in
 962   if(ret < 0){
 963     printf("H5Dread error: Earth Incidence¥n");
 964     exit(1);
 965   }
 966   ret = H5Dclose(dhnd);
 967   // cutoff overlap & change scale
 968   for(j = 0; j < num; ++j){
 969     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
 970       ear_in[j][i] = ear_in[j+ovr][i];
 971       if(ear_in[j][i] > -32767) ear_in[j][i] = ear_in[j][i] * sca;
 972     }
 973
 974   for(j = num; j < LMT; ++j){
 975     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
 976       ear_in[j][i] = 0;
 977     }
 978
 979   // sample display
 980   printf("ear_in[scan=0][pixel=0]: %9.2f¥n",ear_in[0][0]);
```

Read scale factor.

Read earth incidence.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

Remove overlap scans.

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
1008   // close
1009   ret = H5Fclose(fhnd);
1010   ret = H5close();
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

7.1.2 Compile (Explanation of build_readL1B_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL1B_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc="readL1B_hdf5.c amsr2time_.c amsr2latlon_.c"
15
16 # output filename
17 out=readL1B_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL1B_hdf5_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1B_hdf5_c.sh
icc -g readL1B_hdf5.c amsr2time_.c amsr2latlon_.c -o readL1B_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.1.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1B_hdf5_c” is shown as follows.

```
$ ./readL1B_hdf5_c GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGBTBR_0000000
ObservationStartDateTime: 2012-07-26T11:45:43.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:37.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G- 1.25000,7G- 1.00000,10G- 1.25000,18G- 1.25000,23G- 1.25000,36G- 1.00000
CoRegistrationParameterA2: 6G- 0.00000,7G--0.10000,10G--0.25000,18G- 0.00000,23G--0.25000,36G- 0.00000
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leapsec.dat
amsr2time: year=1993 month= 7 tai93sec=  15638401.00
amsr2time: year=1994 month= 7 tai93sec=  47174402.00
amsr2time: year=1996 month= 1 tai93sec=  94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time[scan=0]: 2012/07/26 11:45:43
latlon89a[scan=0][pixel=0]: ( -73.3289, 136.7714)
latlon89b[scan=0][pixel=0]: ( -73.4038, 137.1498)
latlonlm[scan=0][pixel=0]: ( -73.3538, 136.6228)
latlon06[scan=0][pixel=0]: ( -73.3592, 136.6213)
latlon07[scan=0][pixel=0]: ( -73.3497, 136.6429)
latlon10[scan=0][pixel=0]: ( -73.3506, 136.6001)
latlon18[scan=0][pixel=0]: ( -73.3592, 136.6213)
latlon23[scan=0][pixel=0]: ( -73.3506, 136.6001)
latlon36[scan=0][pixel=0]: ( -73.3532, 136.6514)
tb06h[scan=0][pixel=0]:   173.28
tb06v[scan=0][pixel=0]:   208.22
tb07h[scan=0][pixel=0]:   173.07
tb07v[scan=0][pixel=0]:   207.54
tb10h[scan=0][pixel=0]:   170.94
tb10v[scan=0][pixel=0]:   204.95
tb18h[scan=0][pixel=0]:   164.85
tb18v[scan=0][pixel=0]:   199.84
tb23h[scan=0][pixel=0]:   163.22
tb23v[scan=0][pixel=0]:   196.53
tb36h[scan=0][pixel=0]:   156.56
tb36v[scan=0][pixel=0]:   186.39
tb89ah[scan=0][pixel=0]:   163.76
tb89av[scan=0][pixel=0]:   179.27
tb89bh[scan=0][pixel=0]:   170.60
tb89bv[scan=0][pixel=0]:   188.16
pdq06h[scan=0][pixel=0]):  0
pdq06v[scan=0][pixel=0]):  0
pdq07h[scan=0][pixel=0]):  0
pdq07v[scan=0][pixel=0]):  0
pdq10h[scan=0][pixel=0]):  0
pdq10v[scan=0][pixel=0]):  0
pdq18h[scan=0][pixel=0]):  0
pdq18v[scan=0][pixel=0]):  0
pdq23h[scan=0][pixel=0]):  0
pdq23v[scan=0][pixel=0]):  0
pdq36h[scan=0][pixel=0]):  0
pdq36v[scan=0][pixel=0]):  0
pdq89ah[scan=0][pixel=0]):  0
pdq89av[scan=0][pixel=0]):  0
pdq89ah[scan=0][pixel=0]):  0
pdq89av[scan=0][pixel=0]):  0
lof06[scan=0][pixel=0]: 100
lof07[scan=0][pixel=0]: 100
lof10[scan=0][pixel=0]: 100
lof18[scan=0][pixel=0]: 100
lof23[scan=0][pixel=0]: 100
lof36[scan=0][pixel=0]: 100
lof89a[scan=0][pixel=0]: 100
lof89b[scan=0][pixel=0]: 100
ear_in[scan=0][pixel=0]:    55.20
ear_az[scan=0][pixel=0]:   144.76
```

7.2 Read L1R Product

7.2.1 C sample program (readL1R_hdf5.c)

Sample program (readL1R_hdf5.c) which reads the metadata and the datasets of L1R is shown below. Latitude and longitude of low frequency data are extracted from observation point of 89GHz-A. Values of data are dumped to the standard output.

Only part of L1R brightness temperature data are handled in this sample program. Please refer to "3.8 Level1 Resampling Products (L1R)" on page 18.

Subroutines are prepared for the conversion of TAI93.

<table>
<tr><th>Metadata</th><th>Datasets</th></tr>
<tr><td>* GeophysicalName
- GranuleID
- ObservationStartDateTime
- EquatorCrossingDateTime
- ObservationEndDateTime
* NumberOfScans
- OverlapScans
- CoRegistrationParameterA1
- CoRegistrationParameterA2</td><td rowspan="3">* Scan Time
* Latitude of Observation Point for 89A
- Latitude of Observation Point for 89B
- Longitude of Observation Point for 89A
- Longitude of Observation Point for 89B
* Brightness Temperature (res06,6.9GHz,H)
- Brightness Temperature (res06,6.9GHz,V)
- Brightness Temperature (res06,7.3GHz,H)
- Brightness Temperature (res06,7.3GHz,V)
- Brightness Temperature (res10,10.7GHz,H)
- Brightness Temperature (res10,10.7GHz,V)
- Brightness Temperature (res23,18.7GHz,H)
- Brightness Temperature (res23,18.7GHz,V)
- Brightness Temperature (res23,23.8GHz,H)
- Brightness Temperature (res23,23.8GHz,V)
- Brightness Temperature (res36,36.5GHz,H)
- Brightness Temperature (res36,36.5GHz,V)
- Brightness Temperature (res36,89.0GHz,H)
- Brightness Temperature (res36,89.0GHz,V)
- Brightness Temperature (original,89GHz-A,H)
- Brightness Temperature (original,89GHz-A,V)
- Brightness Temperature (original,89GHz-B,H)
- Brightness Temperature (original,89GHz-B,V)
* Pixel Data Quality 6 to 36
- Pixel Data Quality 89
* Land_Ocean Flag 6 to 36
- Land_Ocean Flag 89
* Earth Incidence
- Earth Azimuth</td></tr>
<tr><td>Data extracted from observation point of 89GHz-A</td></tr>
<tr><td>* Latitude and longitude of low frequency data
For details to P.20</td></tr>
</table>

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "hdf5.h"
 4 #include "amsr2time.h"
 :

 6 // fixed value
 7 #define LMT 2200 // limit of NumberOfScans
 8 #define AM2_DEF_SNUM_HI 486 // high resolution pixel width
 9 #define AM2_DEF_SNUM_LO 243 // low  resolution pixel width
10
11 int main(int argc, char *argv[]){
12   // interface variable
13   int i,j;         // loop variable
14   herr_t ret;      // return status
15   char *buf = NULL; // text buffer
16   char *fn  = NULL; // filename
17   hid_t fhnd;      // file handle
18   hid_t ahnd;      // attribute handle
19   hid_t atyp;      // attribute type
20   hid_t dhnd;      // dataset handle
21   hid_t shnd;      // dataspace handle
22
23   // meta data
24   char *geo = NULL; // GeophysicalName
25   char *gid = NULL; // GranuleID
26   char *tm1 = NULL; // ObservationStartDateTime
27   char *tm2 = NULL; // EquatorCrossingDateTime
28   char *tm3 = NULL; // ObservationEndDateTime
29   int num;         // NumberOfScans
30   int ovr;         // OverlapScans
31   double prm1[7];   // CoRegistrationParameterA1
32   double prm2[7];   // CoRegistrationParameterA2

39   // array data
40   AM2_COMMON_SCANTIME st[LMT]; // scantime
41   float lat89ar[LMT][AM2_DEF_SNUM_HI]; // lat for 89a altitude revised
42   float lat89br[LMT][AM2_DEF_SNUM_HI]; // lat for 89b altitude revised
 :
44   float lon89ar[LMT][AM2_DEF_SNUM_HI]; // lon for 89a altitude revised
45   float lon89br[LMT][AM2_DEF_SNUM_HI]; // lon for 89b altitude revised
 :
47   float tb06h06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06h, resolution 06G
48   float tb06v06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06v, resolution 06G
 :
65   unsigned char pdq06h [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06h
66   unsigned char pdq06v [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06v
 :
74   unsigned char lof06 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 06
75   unsigned char lof10 [LMT  ][AM2_DEF_SNUM_LO]; // land ocean flag for 10
 :
81   float ear_in[LMT][AM2_DEF_SNUM_LO]; // earth incidence
82   float ear_az[LMT][AM2_DEF_SNUM_LO]; // earth azimuth
```

Include header file of HDF5 for C language.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
 94   // hdf5 initialize
 95   ret = H5open();
 96   if(ret < 0){
 97     printf("h5open error: %d¥n" ,ret);
 98     exit(1);
 99   }
100
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allows read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
101   // open
102   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
103   if(fhnd < 0){
104     printf("H5Fopen error: %s¥n", fn);
105     exit(1);
106   }
```

Read metadata * Numbers written on the left are row number of the sample program.

First you have to open the attributes which are stored with the data object using H5Aopen function to acquire metadata.
Acquire the attribute type next, and you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
108   // read meta: GeophysicalName
109   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
110   atyp = H5Aget_type(ahnd);
111   ret = H5Aread(ahnd, atyp, &geo);
112   if(ret < 0){
113     printf("H5Aread error: GeophysicalName¥n");
114     exit(1);
115   }
116   ret = H5Aclose(ahnd);
117   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
163   // read meta: NumberOfScans
164   ahnd = H5Aopen(fhnd, "NumberOfScans", H5P_DEFAULT);
165   atyp = H5Aget_type(ahnd);
166   ret = H5Aread(ahnd, atyp, &buf);
167   if(ret < 0){
168     printf("H5Aread error: NumberOfScans¥n");
169     exit(1);
170   }
171   ret = H5Aclose(ahnd);
172   num = atoi(buf);
173   printf("NumberOfScans: %d¥n", num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
Overlap scans contained at both end of L1 data should be removed.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

For details to P.13.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array. Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

```
252   // read array: scantime
253   dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
254   ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
255   if(ret < 0){
256     printf("H5Dread error: Scan Time¥n");
257     exit(1);
258   }
259   ret = H5Dclose(dhnd);
260   // cutoff overlap
261   for(j = 0; j < num; ++j){
262     r8d1[j] = r8d1[j + ovr];
263   }
264   for(j = num; j < LMT; ++j){
265     r8d1[j] = 0;
266   }
267   // convert
268   amsr2time_(&num, r8d1, st);
269   // sample display
270   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
271   , st[0].year
272   , st[0].month
273   , st[0].day
274   , st[0].hour
275   , st[0].minute
276   , st[0].second
277   );
```

Read scanning time.

Remove overlap scans.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude of 89GHz.
Overlap scans contained at both end of L1 data should be removed.
For the latitude and longitude of low frequency data, extract odd sample number of 89GHz-A observation points (origin 1).
* In case of C language, origin is 0 and even sample number of data are extracted.

For details to P.20.

```
279   // read array: latlon for 89a altitude revised
280   // read lat
281   dhnd = H5Dopen(fhnd ,"Latitude of Observation Point for 89A", H5P_DEFAULT);
282   ret = H5Dread(dhnd,H5T_NATIVE_FLOAT,H5S_ALL, H5S_ALL, H5P_DEFAULT, lat89ar);
283   if(ret < 0){
284     printf("H5Dread error: Latitude of Observation Point for 89A¥n");
285     exit(1);
286   }
287   ret = H5Dclose(dhnd);
288   // read lon
289   dhnd = H5Dopen(fhnd ,"Longitude of Observation Point for 89A", H5P_DEFAULT);
290   ret = H5Dread(dhnd,H5T_NATIVE_FLOAT,H5S_ALL, H5S_ALL, H5P_DEFAULT, lon89ar);
291   if(ret < 0){
292     printf("H5Dread error: Longitude of Observation Point for 89A¥n");
293     exit(1);
294   }
295   ret = H5Dclose(dhnd);
296   // cutoff overlap
297   for(j = 0; j < num; ++j){
298     for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
299       lat89ar[j][i] = lat89ar[j+ovr][i];
300       lon89ar[j][i] = lon89ar[j+ovr][i];
301     }
302   }
303   for(j = num; j < LMT; ++j){
304     for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
305       lat89ar[j][i] = 0;
306       lon89ar[j][i] = 0;
307     }
308   }
  :

345   // read array: latlon for low resolution
346   for(j = 0; j < num; ++j){
347     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
348       latlr[j][i] = lat89ar[j][i * 2];
349       lonlr[j][i] = lon89ar[j][i * 2];
350     }
351   }
```

Read latitude.

Read longitude.

Remove overlap scans.

Extract the position of low frequency data from the observation point of 89GHz-A.

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer(0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value(65534) must be excluded.

```
354   // read array: tb for 06h, resolution 06G
355   dhnd = H5Dopen(fhnd, "Brightness Temperature (res06,6.9GHz,H)", H5P_DEFAULT);
356   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT); //
357   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);       // get scale
358   ret = H5Aclose(ahnd);                               // get scale
359   ret = H5Dread(dhnd,H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, tb06h06);
360   if(ret < 0){
361     printf("H5Dread error: Brightness Temperature (res06,6.9GHz,H)¥n");
362     exit(1);
363   }
364   ret = H5Dclose(dhnd);
365   // cutoff overlap & change scale
366   for(j = 0; j < num; ++j){
367     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
368       tb06h06[j][i] = tb06h06[j+ovr][i];
369       if(tb06h06[j][i] < 65534) tb06h06[j][i] = tb06h06[j][i] * sca;
370     }
371   }
372   for(j = num; j < LMT; ++j){
373     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
374       tb06h06[j][i] = 0;
375     }
376   }
377   // sample display
378   printf("tb06h06[scan=0][pixel=0]: %9.2f¥n", tb06h06[0][0]);
```

Read scale factor.

Read brightness temperature.

Remove overlap scans.

Exclude the missing value (65535) and error value(65534) when scale handling is applied.

Read L1R pixel data quality * Numbers written on the left are row number of the sample program.

"Pixel Data Quality 6 to 36" consists of two 1-byte integer and 12 out of 16 bits shows bit by bit each frequency and polarization.
"Pixel Data Quality 89" consists of one 1-byte integer and 4 out of 8 bits shows bit by bit each frequency and polarization.

```
822   // read array: pixel data quality for low
823   dhnd = H5Dopen(fhnd, "Pixel Data Quality 6 to 36", H5P_DEFAULT);
824   ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, i1d2hi);
825   if(ret < 0){
826     printf("H5Dread error: Pixel Data Quality 6 to 36¥n");
827     exit(1);
828   }
829   ret = H5Dclose(dhnd);
```

Read L1R pixel data quality

"Pixel Data Quality 6 to 36" are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to unsigned char type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
830   // cutoff overlap & separate
831   for(j = 0; j < num; ++j){
832     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
833       pdq06v[j][i]=0;
834       if((i1d2hi[j+ovr][i*2+0] &   1) != 0) pdq06v[j][i]=1;
835       if((i1d2hi[j+ovr][i*2+0] &   2) != 0) pdq06v[j][i]=10;
836       if(((i1d2hi[j+ovr][i*2+0] &   1) != 0) && ((i1d2hi[j+ovr][i*2+0] &   2) !=
0))  pdq06v[j][i]=11;
837       pdq06h[j][i]=0;
838       if((i1d2hi[j+ovr][i*2+0] &   4) != 0) pdq06h[j][i]=1;
839       if((i1d2hi[j+ovr][i*2+0] &   8) != 0) pdq06h[j][i]=10;
840       if(((i1d2hi[j+ovr][i*2+0] &   4) != 0) && ((i1d2hi[j+ovr][i*2+0] &   8) !=
0))  pdq06h[j][i]=11;
841       pdq07v[j][i]=0;
842       if((i1d2hi[j+ovr][i*2+0] &  16) != 0) pdq07v[j][i]=1;
843       if((i1d2hi[j+ovr][i*2+0] &  32) != 0) pdq07v[j][i]=10;
844       if(((i1d2hi[j+ovr][i*2+0] &  16) != 0) && ((i1d2hi[j+ovr][i*2+0] &  32) !=
0))  pdq07v[j][i]=11;
845       pdq07h[j][i]=0;
846       if((i1d2hi[j+ovr][i*2+0] &  64) != 0) pdq07h[j][i]=1;
847       if((i1d2hi[j+ovr][i*2+0] & 128) != 0) pdq07h[j][i]=10;
848       if(((i1d2hi[j+ovr][i*2+0] &  64) != 0) && ((i1d2hi[j+ovr][i*2+0] & 128) !=
0))  pdq07h[j][i]=11;
849     }
850   }
851   for(j = num; j < LMT; ++j){
852     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
853       pdq06h[j][i]=0;
854       pdq06v[j][i]=0;
855       pdq07h[j][i]=0;
856       pdq07v[j][i]=0;
857     }
858   }
```

Split the bit by bit information to each array of frequency and polarization.

Remove overlap scans also applied.

Read L1R land ocean flag * Numbers written on the left are row number of the sample program.

Data type of L1R land ocean flag is 1-byte integer and two dimensional (sample * (scan * channel)).
"Land_Ocean Flag 6 to 36" has 6 frequency channels (6, 10, 23, and 36GHz).
"Land_Ocean Flag 89" has 2 frequency channels (89GHz-A and 89GHz-B).

For details to P.18.

```
896   // read array: land ocean flag for low
897   dhnd = H5Dopen(fhnd, "Land_Ocean Flag 6 to 36", H5P_DEFAULT);
898   ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, loflo);
899   if(ret < 0){
900     printf("H5Dread error: Land_Ocean Flag 6 to 36¥n");
901     exit(1);
902   }
903   ret = H5Dclose(dhnd);
```

Read L1R land ocean flag.

"Land_Ocean Flag 6 to 36" and "Land_Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to unsigned char type two dimensional array which is prepared for each frequency for convenience.

```
904   // separate
905   for(j = 0; j < num+ovr*2; ++j){
906     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
907       lof06[j][i]=loflo[(num+ovr*2)*0+j][i];
908       lof10[j][i]=loflo[(num+ovr*2)*1+j][i];
909       lof23[j][i]=loflo[(num+ovr*2)*2+j][i];
910       lof36[j][i]=loflo[(num+ovr*2)*3+j][i];
911     }
912   }
913   // cutoff overlap
914   for(j = 0; j < num; ++j){
915     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
916       lof06[j][i]=lof06[j+ovr][i];
917       lof10[j][i]=lof10[j+ovr][i];
918       lof23[j][i]=lof23[j+ovr][i];
919       lof36[j][i]=lof36[j+ovr][i];
920     }
921   }
922   for(j = num; j < LMT; ++j){
923     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
924       lof06[j][i]=0;
925       lof10[j][i]=0;
926       lof23[j][i]=0;
927       lof36[j][i]=0;
928     }
929   }
```

Split the data to each frequency.

Remove overlap scans also applied.

Read earth incidence * Numbers written on the left are row number of the sample program.

Earth incidence is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the earth incidence data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
 968   // read array: earth incidence
 969   dhnd = H5Dopen(fhnd, "Earth Incidence", H5P_DEFAULT);
 970   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
 971   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
 972   ret = H5Aclose(ahnd);                              // get scale
 973   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
ear_in);
 974   if(ret < 0){
 975     printf("H5Dread error: Earth Incidence¥n");
 976     exit(1);
 977   }
 978   ret = H5Dclose(dhnd);
 979   // cutoff overlap & change scale
 980   for(j = 0; j < num; ++j){
 981     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
 982       ear_in[j][i] = ear_in[j+ovr][i];
 983       if(ear_in[j][i] > -32767) ear_in[j][i] = ear_in[j][i] * sca;
 984     }
 985   }
 986   for(j = num; j < LMT; ++j){
 987     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
 988       ear_in[j][i] = 0;
 989     }
 990   }
 991   // sample display
 992   printf("ear_in[scan=0][pixel=0]: %9.2f¥n",ear_in[0][0]);
```

Read scale factor.

Read earth incidence.

Exclude the missing value (-32768) and error value(-32767) when scale handling is applied.

Remove overlap scans.

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
1020   // close
1021   ret = H5Fclose(fhnd);
1022   ret = H5close();
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

7.2.2 Compile (Explanation of build_readL1R_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL1R_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc="readL1R_hdf5.c amsr2time_.c "
15
16 # output filename
17 out=readL1R_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL1R_hdf5_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1R_hdf5_c.sh
icc -g readL1R_hdf5.c amsr2time_.c amsr2latlon_.c -o readL1R_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.2.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1R_hdf5_c” is shown as follows.

```
$ ./readL1R_hdf5_c GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGRTBR_0000000
ObservationStartDateTime: 2012-07-26T11:45:43.018Z
EquatorCrossingDateTime: 2012-07-26T12:12:37.848Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G- 0.00000,7G- 0.00000,10G- 0.00000,18G- 0.00000,23G- 0.00000,36G- 0.00000
CoRegistrationParameterA2: 6G- 0.00000,7G- 0.00000,10G- 0.00000,18G- 0.00000,23G- 0.00000,36G- 0.00000
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leapsec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time[scan=0]: 2012/07/26 11:45:43
latlon89ar[scan=0][pixel=0]: ( -73.3581, 136.8432)
latlon89br[scan=0][pixel=0]: ( -73.4328, 137.2216)
latlonlr[scan=0][pixel=0]: ( -73.3581, 136.8432)
tb06h06[scan=0][pixel=0]:   173.41
tb06v06[scan=0][pixel=0]:   208.09
tb07h06[scan=0][pixel=0]:   173.07
tb07v06[scan=0][pixel=0]:   207.11
tb10h10[scan=0][pixel=0]:   170.40
tb10v10[scan=0][pixel=0]:   204.58
tb18h23[scan=0][pixel=0]:   165.83
tb18v23[scan=0][pixel=0]:   199.41
tb23h23[scan=0][pixel=0]:   163.55
tb23v23[scan=0][pixel=0]:   195.90
tb36h36[scan=0][pixel=0]:   153.55
tb36v36[scan=0][pixel=0]:   183.95
tb89h36[scan=0][pixel=0]:   163.86
tb89v36[scan=0][pixel=0]:   181.05
tb89ah[scan=0][pixel=0]:   163.76
tb89av[scan=0][pixel=0]:   179.27
tb89bh[scan=0][pixel=0]:   170.60
tb89bv[scan=0][pixel=0]:   188.16
pdq06h[scan=0][pixel=0]):   0
pdq06v[scan=0][pixel=0]):   0
pdq07h[scan=0][pixel=0]):   0
pdq07v[scan=0][pixel=0]):   0
pdq10h[scan=0][pixel=0]):   0
pdq10v[scan=0][pixel=0]):   0
pdq18h[scan=0][pixel=0]):   0
pdq18v[scan=0][pixel=0]):   0
pdq23h[scan=0][pixel=0]):   0
pdq23v[scan=0][pixel=0]):   0
pdq36h[scan=0][pixel=0]):   0
pdq36v[scan=0][pixel=0]):   0
pdq89ah[scan=0][pixel=0]):   0
pdq89av[scan=0][pixel=0]):   0
pdq89ah[scan=0][pixel=0]):   0
pdq89av[scan=0][pixel=0]):   0
lof06[scan=0][pixel=0]: 100
lof10[scan=0][pixel=0]: 100
lof23[scan=0][pixel=0]: 100
lof36[scan=0][pixel=0]: 100
lof89a[scan=0][pixel=0]: 100
lof89b[scan=0][pixel=0]: 100
ear_in[scan=0][pixel=0]:    55.20
ear_az[scan=0][pixel=0]:   144.76
```

7.3 Read L2 Low Resolution Product

L2 Low Resolution Products are Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW). For Precipitation(PRC) product, please refer to "7.4 Read L2 High Resolution Product" on page 166.

7.3.1 C sample program (readL2L_hdf5.c)

Sample program (readL2L_hdf5.c) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point * Longitude of Observation Point * Geophysical Data * Pixel Data Quality For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "hdf5.h"
 5 #include "amsr2time.h"
 6
 7 // fixed value
 8 #define LMT 2200 // limit of NumberOfScans
 9 #define AM2_DEF_SNUM_HI 486 // high resolution pixel width
10 #define AM2_DEF_SNUM_LO 243 // low  resolution pixel width
 :
13 int main(int argc, char *argv[]){
14   // interface variable
15   int i,j;         // loop variable
16   herr_t ret;      // return status
17   char *buf = NULL; // text buffer
18   char *fn  = NULL; // filename
19   hid_t fhnd;      // file handle
20   hid_t ahnd;      // attribute handle
21   hid_t atyp;      // attribute type
22   hid_t dhnd;      // dataset handle
23
24   // meta data
25   char *geo = NULL; // GeophysicalName
26   char *gid = NULL; // GranuleID
27   char *tm1 = NULL; // ObservationStartDateTime
28   char *tm2 = NULL; // EquatorCrossingDateTime
29   char *tm3 = NULL; // ObservationEndDateTime
30   int num;         // NumberOfScans
31   int ovr;         // OverlapScans

37   // array data
38   AM2_COMMON_SCANTIME st[LMT]; // scantime
39   float lat[LMT][AM2_DEF_SNUM_LO]; // lat
40   float lon[LMT][AM2_DEF_SNUM_LO]; // lon
41   float geo1[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 1
42   float geo2[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 2
43   float geotmp[LMT*2][AM2_DEF_SNUM_LO]; // geophysical data temporary

44   unsigned char pdq1[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 1
45   unsigned char pdq2[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 2
46   unsigned char pdqtmp[LMT*2][AM2_DEF_SNUM_LO]; // pixel data quality temporary
```

Include header file of HDF5 for C language.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200. When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
58   // hdf5 initialize
59   ret = H5open();
60   if(ret < 0){
61     printf("h5open error: %d¥n" ,ret);
62     exit(1);
63   }
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allow read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
65   // open
66   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
67   if(fhnd < 0){
68     printf("H5Fopen error: %s¥n", fn);
69     exit(1);
70   }
```

Read metadata * Numbers written on the left are row number of the sample program.

To acquire metadata, H5Aopen function is first needed to open the attributes which are stored with the data object.
Get the attribute type next, then you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
72   // read meta: GeophysicalName
73   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
74   atyp = H5Aget_type(ahnd);
75   ret = H5Aread(ahnd, atyp, &geo);
76   if(ret < 0){
77     printf("H5Aread error: GeophysicalName¥n");
78     exit(1);
79   }
80   ret = H5Aclose(ahnd);
81   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
140   // read meta: NumberOfScans
141   ahnd = H5Aopen(fhnd, "NumberOfScans", H5P_DEFAULT);
142   atyp = H5Aget_type(ahnd);
143   ret = H5Aread(ahnd, atyp, &buf);
144   if(ret < 0){
145     printf("H5Aread error: NumberOfScans¥n");
146     exit(1);
147   }
148   ret = H5Aclose(ahnd);
149   num = atoi(buf);
150   printf("NumberOfScans: %d¥n", num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
Overlap scans contained at both end of L1 data should be removed.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array. Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

```
173   // read array: scantime
174   dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
175   ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
176   if(ret < 0){
177     printf("H5Dread error: Scan Time¥n");
178     exit(1);
179   }
180   ret = H5Dclose(dhnd);
181   // convert
182   amsr2time_(&num, r8d1, st);
183   // sample display
184   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
185   , st[0].year
186   , st[0].month
187   , st[0].day
188   , st[0].hour
189   , st[0].minute
190   , st[0].second
191   );
```

Read scanning time.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude. → For details to P.20

```
200   // read array: latlon
201   // read lat
202   dhnd = H5Dopen(fhnd ,"Latitude of Observation Point", H5P_DEFAULT);
203   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat);
204   if(ret < 0){
205     printf("H5Dread error: Latitude of Observation Point¥n");
206     exit(1);
207   }
208   ret = H5Dclose(dhnd);
209   // read lon
210   dhnd = H5Dopen(fhnd ,"Longitude of Observation Point", H5P_DEFAULT);
211   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon);
212   if(ret < 0){
213     printf("H5Dread error: Longitude of Observation Point¥n");
214     exit(1);
215   }
216   ret = H5Dclose(dhnd);
217   // sample display
218   printf("latlon[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", lat[0][0], lon[0][0]);
```

Read latitude.

Read longitude.

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the geophysical quantity data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
213   // read array: geophysical data for 1 layer
214   if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
215     dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);
216     // get scale
217     ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
218     ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
219     ret = H5Aclose(ahnd);
220     // read
221     ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, geo1);
222     if(ret < 0){
223       printf("H5Dread error: Geophysical Data¥n");
224       exit(1);
225     }
226     ret = H5Dclose(dhnd);
227     // change scale
228     for(j = 0; j < num; ++j){
229       for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
230         if(geo1[j][i] > -32767) geo1[j][i] = geo1[j][i] * sca;
231       }
232     }
233   }
234
```

Read scale factor.

Read geophysical quantities.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value(-32767) when scale handling is applied.

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
235   // read array: geophysical data for 2 layer
236   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
237     dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);
238     // get scale
239     ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
240     ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
241     ret = H5Aclose(ahnd);
242     // read
243     ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
geotmp);
244     if(ret < 0){
245       printf("H5Dread error: Geophysical Data¥n");
246       exit(1);
247     }
248     ret = H5Dclose(dhnd);
249     // separate & change scale
250     for(j = 0; j < num; ++j){
251       for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
252         geo1[j][i] = geotmp[j*2][i*2+0];
253         geo2[j][i] = geotmp[j*2][i*2+1];
254         if(geo1[j][i] > -32767) geo1[j][i] = geo1[j][i] * sca;
255         if(geo2[j][i] > -32767) geo2[j][i] = geo2[j][i] * sca;
256       }
257     }
258   }
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.
Snow depth product and Sea surface temperature product have two layer of pixel data quality.

```
260   // read array: pixel data quality for 1 layer
261   if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
262     dhnd = H5Dopen(fhnd, "Pixel Data Quality", H5P_DEFAULT);
263     ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, pdq1);
264     if(ret < 0){
265       printf("H5Dread error: Pixel Data Quality¥n");
266       exit(1);
267     }
268     ret = H5Dclose(dhnd);
269   }
270
```

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
271   // read array: pixel data quality for 2 layer
272   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
273     // read
274     dhnd = H5Dopen(fhnd, "Pixel Data Quality", H5P_DEFAULT);
275     ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT,
pdqtmp);
276     if(ret < 0){
277       printf("H5Dread error: Pixel Data Quality¥n");
278       exit(1);
279     }
280     ret = H5Dclose(dhnd);
281     // separate
282     for(j = 0; j < num; ++j){
283       for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
284         pdq1[j][i] = pdqtmp[num*0+j][i];
285         pdq2[j][i] = pdqtmp[num*1+j][i];
286       }
287     }
288   }
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
314   // close
315   ret = H5Fclose(fhnd);
316   ret = H5close();
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

7.3.2 Compile(Explanation of build_readL2L_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL2L_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc="readL2L_hdf5.c amsr2time_.c "
15
16 # output filename
17 out=readL2L_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL2L_hdf5_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2L_hdf5_c.sh
icc -g readL2L_hdf5.c amsr2time_.c -o readL2L_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.3.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2L_hdf5_c” is shown as follows.

```
$ ./readL2L_hdf5_c GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_201303011809_125D_L2SGCLWLA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans: 1972
limit of NumberOfScans = 2200
OverlapScans: 0
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leaps
ec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time[scan=0]: 2013/03/01 18:09:10
latlon[scan=0][pixel=0]: (  84.4574, -78.1076)
geo1[scan=0][pixel=0]: -32767.000 [Kg/m2] (PDQ:112)
```

7.4 Read L2 High Resolution Product

Precipitation (PRC) product is L2 High Resolution Product.

For Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW), please refer to “7.3 Read L2 Low Resolution Product” on page 156.

7.4.1 C sample (readL2H_hdf5.c) program

Sample program (readL2H_hdf5.c) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93.

Metadata	Datasets
* GeophysicalName - GranuleID - ObservationStartDateTime - EquatorCrossingDateTime - ObservationEndDateTime * NumberOfScans - OverlapScans	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B * Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Geophysical Data for 89A - Geophysical Data for 89B * Pixel Data Quality for 89A - Pixel Data Quality for 89B For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "hdf5.h"
 5 #include "amsr2time.h"
 6
 7 // fixed value
 8 #define LMT 2200 // limit of NumberOfScans
 9 #define AM2_DEF_SNUM_HI 486 // high resolution pixel width
10 #define AM2_DEF_SNUM_LO 243 // low  resolution pixel width
 :
13 int main(int argc, char *argv[]){
14   // interface variable
15   int i,j;         // loop variable
16   herr_t ret;      // return status
17   char *buf = NULL; // text buffer
18   char *fn  = NULL; // filename
19   hid_t fhnd;      // file handle
20   hid_t ahnd;      // attribute handle
21   hid_t atyp;      // attribute type
22   hid_t dhnd;      // dataset handle
23
24   // meta data
25   char *geo = NULL; // GeophysicalName
26   char *gid = NULL; // GranuleID
27   char *tm1 = NULL; // ObservationStartDateTime
28   char *tm2 = NULL; // EquatorCrossingDateTime
29   char *tm3 = NULL; // ObservationEndDateTime
30   int num;         // NumberOfScans
31   int ovr;         // OverlapScans

37   // array data
38   AM2_COMMON_SCANTIME st[LMT]; // scantime
39   float lat89a[LMT][AM2_DEF_SNUM_HI]; // lat for 89a
40   float lat89b[LMT][AM2_DEF_SNUM_HI]; // lat for 89b
41   float lon89a[LMT][AM2_DEF_SNUM_HI]; // lon for 89a
42   float lon89b[LMT][AM2_DEF_SNUM_HI]; // lon for 89b
43   float geo1_89a[LMT][AM2_DEF_SNUM_HI];
44   float geo1_89b[LMT][AM2_DEF_SNUM_HI];
45   unsigned char pdq1_89a[LMT][AM2_DEF_SNUM_HI];
46   unsigned char pdq1_89b[LMT][AM2_DEF_SNUM_HI];
```

Include header file of HDF5 for C language.

Include header files of subroutines for time conversion and position calculation.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
58   // hdf5 initialize
59   ret = H5open();
60   if(ret < 0){
61     printf("h5open error: %d¥n" ,ret);
62     exit(1);
63   }
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allow read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
65   // open
66   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
67   if(fhnd < 0){
68     printf("H5Fopen error: %s¥n", fn);
69     exit(1);
70   }
```

Read metadata * Numbers written on the left are row number of the sample program.

To acquire metadata, H5Aopen function is first needed to open the attributes which are stored with the data object.
Get the attribute type next, then you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
72   // read meta: GeophysicalName
73   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
74   atyp = H5Aget_type(ahnd);
75   ret = H5Aread(ahnd, atyp, &geo);
76   if(ret < 0){
77     printf("H5Aread error: GeophysicalName¥n");
78     exit(1);
79   }
80   ret = H5Aclose(ahnd);
81   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read the number of scans from metadata.
Number of scan is necessary when reading datasets.

```
133   // read meta: NumberOfScans
134   ahnd = H5Aopen(fhnd, "NumberOfScans", H5P_DEFAULT);
135   atyp = H5Aget_type(ahnd);
136   ret = H5Aread(ahnd, atyp, &buf);
137   if(ret < 0){
138     printf("H5Aread error: NumberOfScans¥n");
139     exit(1);
140   }
141   ret = H5Aclose(ahnd);
142   num = atoi(buf);
143   printf("NumberOfScans: %d¥n", num);
```

All values of metadata are acquired as character type, so you need to convert the value to numerical.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array.
Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

```
166   // read array: scantime
      dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
      ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
      if(ret < 0){
        printf("H5Dread error: Scan Time¥n");
        exit(1);
      }
      ret = H5Dclose(dhnd);
      // convert
      amsr2time_(&num, r8d1, st);
      // sample display
177   printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d¥n"
178   , st[0].year
179   , st[0].month
180   , st[0].day
181   , st[0].hour
182   , st[0].minute
183   , st[0].second
184   );
```

Read scanning time.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude.

For details to P.20

```
186   // read array: latlon for 89a
187   // read lat
188   dhnd = H5Dopen(fhnd ,"Latitude of Observation Point for 89A", H5P_DEFAULT);
189   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat89a);
190   if(ret < 0){
191     printf("H5Dread error: Latitude of Observation Point for 89A¥n");
192     exit(1);
193   }
194   ret = H5Dclose(dhnd);
195   // read lon
196   dhnd = H5Dopen(fhnd ,"Longitude of Observation Point for 89A", H5P_DEFAULT);
197   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon89a);
198   if(ret < 0){
199     printf("H5Dread error: Longitude of Observation Point for 89A¥n");
200     exit(1);
201   }
202   ret = H5Dclose(dhnd);
203   // sample display
204   printf("latlon89a[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", lat89a[0][0],
lon89a[0][0]);
```

Read latitude.

Read longitude.

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
226   // read array: geophysical data for 1 layer for 89a
227   dhnd = H5Dopen(fhnd, "Geophysical Data for 89A", H5P_DEFAULT);
228   // get scale
229   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
230   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
231   ret = H5Aclose(ahnd);
232   // read
233   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
geo1_89a);
234   if(ret < 0){
235     printf("H5Dread error: Geophysical Data for 89A¥n");
236     exit(1);
237   }
238   ret = H5Dclose(dhnd);
239   // change scale
240   for(j = 0; j < num; ++j){
241     for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
242       if(geo1_89a[j][i] > -32767) geo1_89a[j][i] = geo1_89a[j][i] * sca;
243     }
244   }
```

Read scale factor.

Read geophysical quantity.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.

```
266   // read array: pixel data quality for 1 layer for 89a
267   dhnd = H5Dopen(fhnd, "Pixel Data Quality for 89A", H5P_DEFAULT);
268   ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT,
pdq1_89a);
269   if(ret < 0){
270     printf("H5Dread error: Pixel Data Quality for 89A¥n");
271     exit(1);
272   }
273   ret = H5Dclose(dhnd);
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
314   // close
315   ret = H5Fclose(fhnd);
316   ret = H5close();
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

7.4.2 Compile (Explanation of build_readL2H_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL2H_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc="readL2H_hdf5.c amsr2time_.c "
15
16 # output filename
17 out=readL2H_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL2H_hdf5_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2H_hdf5_c.sh
icc -g readL2H_hdf5.c amsr2time_.c -o readL2H_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.4.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2H_hdf5_c” is shown as follows.

```
$ ./readL2H_hdf5_c GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
GeophysicalName: Precipitation
GranuleID: GW1AM2_201303011809_125D_L2SGPRCHA0000000
ObservationStartDateTime: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDateTime: 2013-03-01T18:58:26.342Z
NumberOfScans: 1972
limit of NumberOfScans = 2200
OverlapScans: 0
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leaps
ec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time[scan=0]: 2013/03/01 18:09:10
latlon89a[scan=0][pixel=0]: (  84.4188, -77.9502)
latlon89b[scan=0][pixel=0]: (  84.3305, -78.8925)
geo1_89a[scan=0][pixel=0]: -32767.0 [mm/h] (PDQ: 16)
geo1_89b[scan=0][pixel=0]: -32767.0 [mm/h] (PDQ: 16)
```

7.5 Read L3 Product [Brightness temperature]

7.5.1 C sample program (readL3B_hdf5.c)

Sample program (readL3B_hdf5.c) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Brightness Temperature (H) - Brightness Temperature (V)

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "hdf5.h"
 :
 9 int main(int argc, char *argv[]){
10   // interface variable
11   int i,j;         // loop variable
12   herr_t ret;      // return status
13   char *buf = NULL; // text buffer
14   char *fn  = NULL; // filename
15   hid_t fhnd;      // file handle
16   hid_t ahnd;      // attribute handle
17   hid_t atyp;      // attribute type
18   hid_t dhnd;      // dataset handle
19   hid_t shnd;      // dataspace handle
20   hsize_t sz1[3];   // array size 1
21   hsize_t sz2[3];   // array size 2
22   int x;           // grid size x
23   int y;           // grid size y
 :
31   // meta data
32   char *geo = NULL; // GeophysicalName
33   char *gid = NULL; // GranuleID
34
 :
38   // array data
39   float *tbH; // brightness temperature for horizontal
40   float *tbV; // brightness temperature for vertical
```

Include header file of HDF5 for C language.

Define interface variables for HDF5.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
52   // hdf5 initialize
53   ret = H5open();
54   if(ret < 0){
55     printf("h5open error: %d¥n" ,ret);
56     exit(1);
57   }
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allow read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
59   // open
60   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
61   if(fhnd < 0){
62     printf("H5Fopen error: %s¥n", fn);
63     exit(1);
64   }
```

Read metadata * Numbers written on the left are row number of the sample program.

To acquire metadata, H5Aopen function is first needed to open the attributes which are stored with the data object.
Get the attribute type next, then you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
66   // read meta: GeophysicalName
67   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
68   atyp = H5Aget_type(ahnd);
69   ret = H5Aread(ahnd, atyp, &geo);
70   if(ret < 0){
71     printf("H5Aread error: GeophysicalName¥n");
72     exit(1);
73   }
74   ret = H5Aclose(ahnd);
75   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
shnd = H5Dget_space(dhnd)
dhnd: Dataset id
shnd: [Return value] Normal: Dataspace id is returned. Error: A negative value is returned.

Acquisition of data array dimension.
ret = H5Sget_simple_extent_dims(shnd,sz1,sz2)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
101   // get grid size
102   dhnd = H5Dopen(fhnd ,"Brightness Temperature (H)", H5P_DEFAULT);
103   shnd = H5Dget_space(dhnd);
104   ret = H5Sget_simple_extent_dims(shnd,sz1,sz2);
105   if(ret < 0){
106     printf("H5Sget_simple_extent_dims error: Brightness Temperature (H)¥n");
107     exit(1);
108   }
109   ret = H5Sclose(shnd);
110   ret = H5Dclose(dhnd);
111   x=sz1[1];
112   y=sz1[0];
113   printf("grid size x: %d¥n", x);
114   printf("grid size y: %d¥n", y);
115
```

Release the dataspace.
ret = H5Sclose(shnd)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Allocate the memory.

```
116   // memory allocate
117   tbH=malloc(sizeof(float)*x*y);
118   if(tbH==NULL){
119     printf("memory allocate error: tbH¥n");
120     exit(1);
121   }
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer (0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value(65534) must be excluded.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array.
Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

```
128   // read horizontal
129   dhnd = H5Dopen(fhnd, "Brightness Temperature (H)", H5P_DEFAULT);
130   // get scale
131   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
132   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
133   ret = H5Aclose(ahnd);
134   // read
      ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, tbH);
      if(ret < 0){
        printf("H5Dread error: Brightness Temperature (H)¥n");
        exit(1);
      }
      ret = H5Dclose(dhnd);
      // change scale
      for(j = 0; j < y; ++j){
        for(i = 0; i < x; ++i){
          if(tbH[x*j+i] < 65534) tbH[x*j+i] = tbH[x*j+i] * sca;
        }
      }
```

Read scale factor.

Read brightness temperature.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (65535) and error value(65534) when scale handling is applied.

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
252   // memory free
253   free(tbH);
254   free(tbV);
```

Close the HDF5 file.

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

```
256   // close
257   ret = H5Fclose(fhnd);
258   ret = H5close();
```

7.5.2 Compile (Explanation of build_readL3B_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL3B_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc=readL3B_hdf5.c
15
16 # output filename
17 out=readL3B_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL3B_hdf5_c.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3B_hdf5_c.sh
icc -g readL3B_hdf5.c -o readL3B_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.5.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

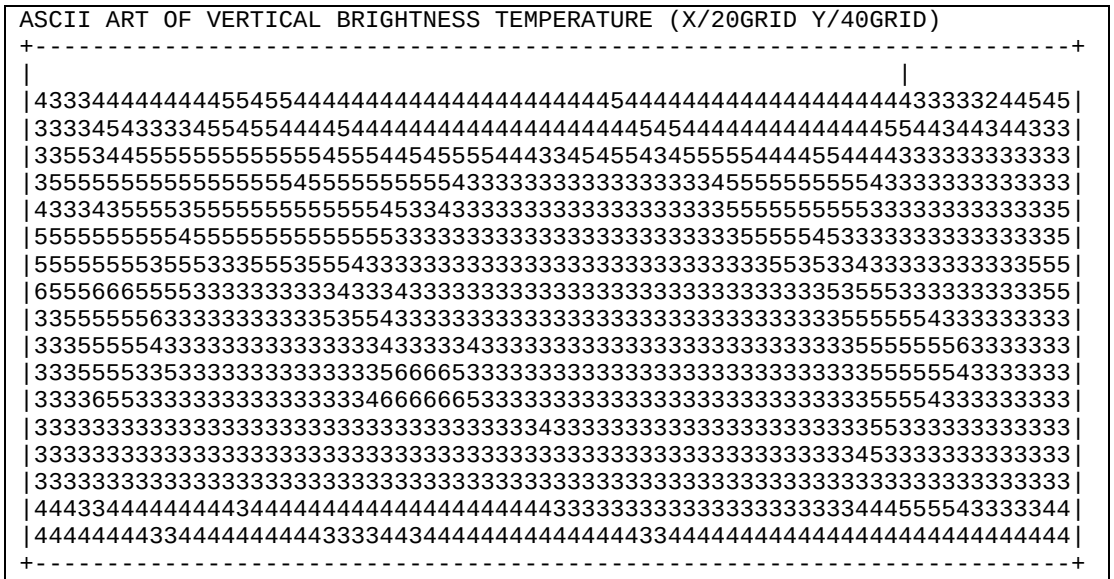
$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3B_hdf5_c” is shown as follows.

```
$ ./readL3B_hdf5_c GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
GeophysicalName: Brightness Temperature (6GHz)
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110
grid size x: 1440
grid size y: 720

ASCII ART OF HORIZONTAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|411233344443444444444444444444444444444444444444444444433443222222444 4|
|111134311113444444444444444444444444444444444444444444444444434233343111|
|124523445555555444444444444444434312343444144555444444444442112111111111|
|154544444444444444444444444544111111111111111111355444444444222111111112|
|412242554414445444444453411311111111111111111114555444455111111111111115|
|444444444244455544555544111111111111111111111115555535111111111111111124|
|444444424441115551555311111111111111111121111111155151141111111111111444|
|555555555551111211113111211111111111111111111111111111141554111111111155|
|115555555111111111115155311111111111111111111111111111145555541111111111|
|111555553111111111111111421111312111111111111111111111155555555111111111|
|111555521511111111111111455555111111111111111111111111145555521111111111|
|111155511111111111111135555555111111111111111111111111154552111111111111|
|111111111111111111111111111111111131111111111111111111155111111111111111|
|111111111111111111111111111111111111111111111111111111351111111111111111|
|111111111111111111111111111111111111111111111111111111111111111111111111|
|343323444333342434334344444444334332111111111111111111344444441111133|
|333333333333333333333333333333343333333222223333334433443333333333333333|
+------------------------------------------------------------------------+
```

ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)



[#]:missing
[ ]:out of observation
[1]: 50-100K
[2]:100-150K
[3]:150-200K
[4]:200-250K
[5]:250-300K
[6]:300-350K
[*]:other

7.6 Read L3 Product [Geophysical quantity]

7.6.1 C sample program (readL3G_hdf5.c)

Sample program (readL3G_hdf5.c) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* GeophysicalName - GranuleID	* Geophysical Data

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include "hdf5.h"
 :
 9 int main(int argc, char *argv[]){
10   // interface variable
11   int i,j;         // loop variable
12   herr_t ret;      // return status
13   char *buf = NULL; // text buffer
14   char *fn  = NULL; // filename
15   hid_t fhnd;      // file handle
16   hid_t ahnd;      // attribute handle
17   hid_t atyp;      // attribute type
18   hid_t dhnd;      // dataset handle
19   hid_t shnd;      // dataspace handle
20   hsize_t sz1[3];   // array size 1
21   hsize_t sz2[3];   // array size 2
22   int x;           // grid size x
23   int y;           // grid size y
 :
33   // meta data
34   char *geo = NULL; // GeophysicalName
35   char *gid = NULL; // GranuleID
 :
40   // array data
41   float *geo1; // geophysical data layer 1
42   float *geo2; // geophysical data layer 2
43   float *geotmp; // geophysical data temporary
```

Include header file of HDF5 for C language.

Define interface variables for HDF5.

Define the variables for metadata.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

```
55   // hdf5 initialize
56   ret = H5open();
57   if(ret < 0){
58     printf("h5open error: %d¥n" ,ret);
59     exit(1);
60   }
```

Open the HDF5 file.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, label1, label2);
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY allow read-only access to file.
label2: Use H5P_DEFAULT for default file access properties.
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

```
62   // open
63   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
64   if(fhnd < 0){
65     printf("H5Fopen error: %s¥n", fn);
66     exit(1);
67   }
```

Read metadata * Numbers written on the left are row number of the sample program.

To acquire metadata, H5Aopen function is first needed to open the attributes which are stored with the data object.
Get the attribute type next, then you'll be able to read the metadata.
Close the attribute identifier after you read the data.

Open an attribute.
ahnd = H5Aopen(fhnd, nam, label);
fhnd: Object id.
nam: Name of attribute.
label: Use H5P_DEFAULT.
ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.

Get an attribute data type.
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.

Read an attribute.
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
ret: [Return value] Error: A negative value is returned.

```
69   // read meta: GeophysicalName
70   ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
71   atyp = H5Aget_type(ahnd);
72   ret = H5Aread(ahnd, atyp, &geo);
73   if(ret < 0){
74     printf("H5Aread error: GeophysicalName¥n");
75     exit(1);
76   }
77   ret = H5Aclose(ahnd);
78   printf("GeophysicalName: %s¥n", geo);
```

Close the attribute.
ret = H5Aclose(ahnd);
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.

Open an existing dataset.
dhnd = H5Dopen(fhnd, nam, label);
fhnd: HDF access file id.
nam: The name of the dataset to access.
label: Use H5P_DEFAULT.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
shnd = H5Dget_space(dhnd)
dhnd: Dataset id
shnd: [Return value] Normal: Dataspace id is returned. Error: A negative value is returned.

Acquisition of data array dimension.
ret = H5Sget_simple_extent_dims(shnd,sz1,sz2)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
105   // get grid size
106   dhnd = H5Dopen(fhnd ,"Geophysical Data", H5P_DEFAULT);
107   shnd = H5Dget_space(dhnd);
108   ret = H5Sget_simple_extent_dims(shnd,sz1,sz2);
109   if(ret < 0){
110     printf("H5Sget_simple_extent_dims error: Geophysical Data¥n");
111     exit(1);
112   }
113   ret = H5Sclose(shnd);
114   ret = H5Dclose(dhnd);
115   x=sz1[1];
116   y=sz1[0];
117   printf("grid size x: %d¥n", x);
118   printf("grid size y: %d¥n", y);
```

Release the dataspace.
ret = H5Sclose(shnd)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
ret = H5Dclose(dhnd);
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Allocate the memory.

```
120   // memory allocate layer 1
121   geo1=malloc(sizeof(float)*x*y);
122   if(geo1==NULL){
123     printf("memory allocate error: geo1¥n");
124     exit(1);
125   }
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

Read raw data from a dataset.
ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);
dhnd: Dataset id.
otyp: Datatype id. (Dataset is automatically converted to specified data type.)
label1,label2: Use when reading part of the array. Use H5S_ALL for both labels when reading entire array.
label3: Use H5P_DEFAULT.
buf: Variable storing acquired data.
ret: [Return value] Error: A negative value is returned.

```
141   // read layer 1
142   if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
143     dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);
144     // get scale
145     ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
146     ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
147     ret = H5Aclose(ahnd);
148     // read
      ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, geo1);
      if(ret < 0){
        printf("H5Dread error: Geophysical Data¥n");
        exit(1);
      }
      ret = H5Dclose(dhnd);
155   // change scale
      for(j = 0; j < y; ++j){
        for(i = 0; i < x; ++i){
          if(geo1[x*j+i] > -32767) geo1[x*j+i] = geo1[x*j+i] * sca;
        }
      }
161   }
```

Read scale factor.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value(-32767) when scale handling is applied.

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
163   // read layer 1 & 2
164   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
165     dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);
166     // get scale
167     ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
168     ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
169     ret = H5Aclose(ahnd);
170     // read
171     ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
geotmp);
172     if(ret < 0){
173       printf("H5Dread error: Geophysical Data¥n");
174       exit(1);
175     }
176     ret = H5Dclose(dhnd);
177     // separate
178     for(j = 0; j < y; ++j){
179       for(i = 0; i < x; ++i){
180         geo1[x*j+i] = geotmp[x*j*2+i*2+0];
181         geo2[x*j+i] = geotmp[x*j*2+i*2+1];
182       }
183     }
184     // change scale
185     for(j = 0; j < y; ++j){
186       for(i = 0; i < x; ++i){
187         if(geo1[x*j+i] > -32767) geo1[x*j+i] = geo1[x*j+i] * sca;
188         if(geo2[x*j+i] > -32767) geo2[x*j+i] = geo2[x*j+i] * sca;
189       }
190     }
191   }
```

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
373   // memory free
374   free(geo1);
375   if(strncmp(gid+29,"SND",3)==0 || strncmp(gid+29,"SST",3)==0){
376     free(geo2);
377   }
```

Close the HDF5 file.

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

```
379   // close
380   ret = H5Fclose(fhnd);
381   ret = H5close();
```

7.6.2 Compile (Explanation of build_readL3G_hdf5_c.sh)

We explain how to compile the C program by using script “build_readL3G_hdf5_c.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.

```
10 # compiler
11 cc=icc
12
13 # source filename
14 csrc=readL3G_hdf5.c
15
16 # output filename
17 out=readL3G_hdf5_c
18
19 # library order
20 lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
21
22 # c compile
23 cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL3G_hdf5_c.sh” is shown in the following. * Line feeds are inserted for convenience.

```
$ ./build_readL3G_hdf5_c.sh
icc -g readL3G_hdf5.c -o readL3G_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
~
```

7.6.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3G_hdf5_c” is shown as follows.

```
$ ./readL3G_hdf5_c GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000
grid size x: 1440
grid size y: 720

ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/20GRID Y/40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|#21#####################################################################|
|2212###1113#############################################################222|
|1#################################11######3###############011#42223322|
|2#############################00101112413333233##############343413131#|
|#20#######2###############00#12234523442132111 20#########3355421131121#|
|#########################4#32222232321001101111 10#######14*41111201111##|
|########0###000###0####1111111210120111 0*131121100##1#11#01211110100###|
|###########1011111111#211#11111210011121333223224321000#1###1111000000##|
|11#######00010122125#3###1#12232321100010020110001011122#######222321111|
|000######122131112311112#01112#113122123231101000000000110########1001110|
|100####32#12111141111000######223232242121244133111001101 0######11111100|
|1000###121123221125110 0########131112232110121313122011010#####231111001|
|11131212231110111111211111110012102#123122122331121311102 1##011124223242|
|*33222511232222222232233314210224223222110242222222122113##0111421221132|
|10112211222111122211111111122231223223233124332121232322122331223222 2111|
|#######################################3111221222211121122 1########21221##|
|#########################################################################|
+------------------------------------------------------------------------+

[#]:missing
[ ]:out of observation
[0]:  0.000 -  0.030 Kg/m2
[1]:  0.030 -  0.060 Kg/m2
[2]:  0.060 -  0.090 Kg/m2
[3]:  0.090 -  0.120 Kg/m2
[4]:  0.120 -  0.150 Kg/m2
[5]:  0.150 -  0.180 Kg/m2
[*]:other
```

8. FORTRAN90 Programming with HDF5 library (without AMTK)

Letters written in red are explanation of the sample programs

Letters written in blue are explanation of the functions used in the sample programs or basic information of the GCOM-W1 satellite and AMSR2 instrument.

8.1 Read L1B Product

8.1.1 FORTRAN sample program (readL1B_hdf5.f)

Sample program (readL1B_hdf5.f) which reads the metadata and the datasets of L1B is shown below. Latitude and longitude of low frequency data are also calculated using observation point of 89GHz-A. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93 and calculation of low frequency latitude and longitude.

<table>
<tr><th>Metadata</th><th>Datasets</th></tr>
<tr><td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.

* NumberOfScans are acquired from the array dimension of the dataset.

Fixed values are used for the variables below.
* OverlapScans
* CoRegistrationParameterA1
* CoRegistrationParameterA2</td><td rowspan="3">* Scan Time
* Latitude of Observation Point for 89A
- Latitude of Observation Point for 89B
- Longitude of Observation Point for 89A
- Longitude of Observation Point for 89B
* Brightness Temperature (6.9GHz,H)
- Brightness Temperature (6.9GHz,V)
- Brightness Temperature (7.3GHz,H)
- Brightness Temperature (7.3GHz,V)
- Brightness Temperature (10.7GHz,H)
- Brightness Temperature (10.7GHz,V)
- Brightness Temperature (18.7GHz,H)
- Brightness Temperature (18.7GHz,V)
- Brightness Temperature (23.8GHz,H)
- Brightness Temperature (23.8GHz,V)
- Brightness Temperature (36.5GHz,H)
- Brightness Temperature (36.5GHz,V)
- Brightness Temperature (89.0GHz-A,H)
- Brightness Temperature (89.0GHz-A,V)
- Brightness Temperature (89.0GHz-B,H)
- Brightness Temperature (89.0GHz-B,V)
* Pixel Data Quality 6 to 36
- Pixel Data Quality 89
* Land_Ocean Flag 6 to 36
- Land_Ocean Flag 89
* Earth Incidence
- Earth Azimuth</td></tr>
<tr><td>Data calculated from observation point of 89GHz-A</td></tr>
<tr><td>* Latitude and longitude (Low mean)
- Latitude and longitude (6G)
- Latitude and longitude (7G)
- Latitude and longitude (10G)
- Latitude and longitude (18G)
- Latitude and longitude (23G)
- Latitude and longitude (36G)
→ For details to P.20</td></tr>
</table>

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 4 C include
 5      include 'amsr2time_f.h'
 :
 8 C fixed value
 9      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
10      integer(4),parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
11      integer(4),parameter::AM2_DEF_SNUM_LO=243 ! low  resolution pixel width
12 C interface variable
13      integer(4) i,j          ! loop variable
14      integer(4) ret          ! return status
15      character(len=512) buf  ! text buffer
16      character(len=512) fn   ! filename
17      integer(HID_T) fhnd     ! file handle
18      integer(HID_T) ahnd     ! attribute handle
19      integer(HID_T) atyp     ! attribute type
20      integer(HID_T) dhnd     ! dataset handle
21      integer(HID_T) dtyp     ! dataset type
22      integer(HID_T) shnd     ! dataspace handle
23      integer(HSIZE_T) sz1(3) ! array size 1
24      integer(HSIZE_T) sz2(3) ! array size 2
 :
35      integer(4) num          ! NumberOfScans
36      integer(4) ovr          ! OverlapScans
37      real(8) prm1(7)         ! CoRegistrationParameterA1
38      real(8) prm2(7)         ! CoRegistrationParameterA2
39      parameter(ovr=20)
 :
45      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
46      real(4) lat89a(AM2_DEF_SNUM_HI,LMT) ! lat for 89a
47      real(4) lat89b(AM2_DEF_SNUM_HI,LMT) ! lat for 89b
 :
55      real(4) lon89a(AM2_DEF_SNUM_HI,LMT) ! lon for 89a
56      real(4) lon89b(AM2_DEF_SNUM_HI,LMT) ! lon for 89b
 :
64      real(4) tb06h(AM2_DEF_SNUM_LO,LMT)  ! tb for 06h
65      real(4) tb06v(AM2_DEF_SNUM_LO,LMT)  ! tb for 06v
 :
80      integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT)  ! pixel data quality for 06h
81      integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT)  ! pixel data quality for 06v
 :
89      integer(1) lof06(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 06
90      integer(1) lof07(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 07
 :
98      real(4) ear_in(AM2_DEF_SNUM_LO,LMT) ! earth incidence
99      real(4) ear_az(AM2_DEF_SNUM_LO,LMT) ! earth azimuth
```

Employ the USE statement to make use of the HDF5 functions in the program.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.
※Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. Fixed values are set in the program.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
108 C hdf5 initialize
109       call H5open_f(ret)
110       if(ret.lt.0)then
111         write(*,'(a,i12)')'h5open_f error: ',ret
112         call exit(1)
113       endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allows read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
114 C open
115       call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
116       if(ret.lt.0)then
117         write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
118         call exit(1)
119       endif
```

Specify NumberOfScans and CoRegistrationParameters * Numbers written on the left are row number of the sample program.

Since the metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library, you need to acquire “NumberOfScans” from array dimension of the dataset “Scan Time”. First, open the dataset by using “H5Dopen_f” function. Next, get the dataspace type by using “H5Dget_space_f” function. Then you'll be able to acquire the array dimension of the dataset by using “H5Sget_simple_extent_dims_f” function.Close the dataset and dataspace identifier after you read the data.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension.
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
120 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
121 C calculate NumberOfScans from array size and OverlapScans
122       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
123       call H5Dget_space_f(dhnd,shnd,ret)
124       call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
125       if(ret.lt.0)then
126         write(*,'(a)')'H5Sget_simple_extent_dims error: Scan Time'
127         call exit(1)
128       endif
129       call H5Sclose_f(shnd,ret)
130       call H5Dclose_f(dhnd,ret)
131       num=sz1(1)-ovr*2
132       write(*,'(a,i12)')'NumberOfScans(RETRIEVE BY ARRAY SIZE): ',num
133       write(*,'(a,i12)')'OverlapScans(FIXED VALUE): ',ovr
```

NumberOfScans is represented as below.
Sum of scans in product - (OverlapScans x 2)

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Specify the CoRegistrationParameters.
These parameters are required when calculating latitudes and longitudes of low frequency data.

```
141 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
142 C use fixed value for CoRegistrationParameter
143       prm1(1)= 1.25000D0
144       prm1(2)= 1.00000D0
145       prm1(3)= 1.25000D0
146       prm1(4)= 1.25000D0
147       prm1(5)= 1.25000D0
148       prm1(6)= 1.00000D0
149       prm1(7)=(prm1(1)+prm1(2)+prm1(3)+prm1(4)+prm1(5)+prm1(6))/6.0D0
:
154       prm2(1)= 0.00000D0
155       prm2(2)=-0.10000D0
156       prm2(3)=-0.25000D0
157       prm2(4)= 0.00000D0
```

For details to P.20.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
Dimension size of the dataset are required when reading.
Overlap scans contained at both end of L1 data should be removed.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

For details to P.13.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
165 C read array: scantime
166       ! read
167       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
168       sz1(1)=LMT
169       call H5Dread_f(dhnd
170      +,H5T_NATIVE_DOUBLE,r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
171       if(ret.lt.0)then
172         write(*,'(a,a)')'H5Dread error: ','
173         call exit(1)
174       endif
175       call H5Dclose_f(dhnd,ret)
176       ! cutoff overlap
177       do j=1,num
178         r8d1(j)=r8d1(j+ovr)
179       enddo
180       do j=num+1,LMT
181         r8d1(j)=0
182       enddo
183       ! convert
184       call amsr2time(num,r8d1,s
185       ! sample display
186       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
187      +'time(scan=1): '
188      +,st(1)%year
189      +,st(1)%month
190      +,st(1)%day
191      +,st(1)%hour
192      +,st(1)%minute
193      +,st(1)%second
```

Read scanning time.

Remove overlap scans.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude of 89GHz.
Overlap scans contained at both end of L1 data should be removed.
Latitudes and longitudes of low frequency data are calculated from position of 89GHz-A and the co-registration parameters.

```
194 C read array: latlon for 89a
195       ! read lat
196       call H5Dopen_f(fhnd
197      +,'Latitude of Observation Point for 89A',dhnd,ret)
198       sz1(1)=LMT
199       sz1(2)=AM2_DEF_SNUM_HI
200       call H5Dread_f(dhnd
201      +,H5T_NATIVE_REAL,lat89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
202       if(ret.lt.0)then
203         write(*,'(a,a)')'H5Dread error: '
204      +  ,'Latitude of Observation Point for 89A'
205         call exit(1)
206       endif
207       call H5Dclose_f(dhnd,ret)
208       ! read lon
209       call H5Dopen_f(fhnd
210      +,'Longitude of Observation Point for 89A',dhnd,ret)
211       sz1(1)=LMT
212       sz1(2)=AM2_DEF_SNUM_HI
213       call H5Dread_f(dhnd
214      +,H5T_NATIVE_REAL,lon89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
215       if(ret.lt.0)then
216         write(*,'(a,a)')'H5Dread error: '
217      +  ,'Longitude of Observation Point for 89A'
218         call exit(1)
219       endif
220       call H5Dclose_f(dhnd,ret)
221       ! cutoff overlap
222       do j=1,num
223         lat89a(:,j)=lat89a(:,j+ovr)
224         lon89a(:,j)=lon89a(:,j+ovr)
225       enddo
226       do j=num+1,LMT
227         lat89a(:,j)=0
228         lon89a(:,j)=0
229       enddo
230       ! sample display
231       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon89a(pixel=1,scan=1): '
232      +,lat89a(1,1),lon89a(1,1)
:
272 C read array: la
273       call amsr2latlon(num,prm1(7),prm2(7),lat89a,lon89a,latlm,lonlm)
274       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlonlm(pixel=1,scan=1): '
 275      +,latlm(1,1),lonlm(1,1)
```

Read latitude.

Read longitude.

Remove overlap scans.

AMSR2 has frequency channels of 6, 7, 10, 18, 23, 36, and 89GHz and their observation point are not strictly same.
Latitude and longitude stored in L1B products are that of 89GHz. Position of low frequency data can be calculated from position of 89GHz-A and the co-registration parameters.
Subroutine for calculating the position of low frequency data is prepared in sample programs. This calculation is automatically applied when using AMTK. For details to P.20.

Calculate the position of low frequency data.

Calculating the position of low frequency data.
call amsr2latlon_(num,prm1,prm2,lat89a,lon89a,latlow,lonlow)
num: Number of scan.
prm1: CoRegistrationParameterA1 for each frequency(6G/7G/10G/18G/23G/36G/Low mean)
prm2: CoRegistrationParameterA2 for each frequency(6G/7G/10G/18G/23G/36G/Low mean)
lat89a: Latitude of 89GHz-A.
lon89a: Longitude of 89GHz-A.
latlow: [Return value] Latitude of specified frequency.
lonlow: [Return value] Longitude of specified frequency.

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer (0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value (65534) must be excluded.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

```
300 C read array: tb for 06h
301       ! read
302       call H5Dopen_f(fhnd
303      +,'Brightness Temperature (6.9GHz,H)',dhnd,ret)
304       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
305       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
306       call H5Aclose_f(ahnd,ret)                    ! get scale
307       sz1(1)=LMT
308       sz1(2)=AM2_DEF_SNUM_LO
309       call H5Dread_f(dhnd
310      +,H5T_NATIVE_REAL,tb06h,sz1,ret,H5S_ALL_F,H5S_ALL_F)
311       if(ret.lt.0)then
312         write(*,'(a,a)')'H5Dread error:
313      +  ,'Brightness Temperature (6.9GHz,H)'
314         call exit(1)
315       endif
316       call H5Dclose_f(dhnd,ret)
317       ! cutoff overlap & convert to unsignd & change scale
318       do j=1,num
319         do i=1,AM2_DEF_SNUM_LO
320           tb06h(i,j)=tb06h(i,j+ovr)
321           if(tb06h(i,j).lt.65534)tb06h(i,j)=tb06h(i,j)*sca
322         enddo
323
324       do j=num+1,LMT
325         tb06h(:,j)=0
326       enddo
327       ! sample display
 328       write(*,'(a,f9.2)')'tb06h(pixel=1,scan=1): ',tb06h(1,1)
```

Read scale factor.

Read brightness temperature.

Remove overlap scans.

Exclude the missing value (65535) and error value(65534) when scale handling is applied.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Scale handling is automatically applied when using AMTK. For details to P.22.

Read L1B pixel data quality * Numbers written on the left are row number of the sample program.

"Pixel Data Quality 6 to 36" consist of two 1-byte integer and 12 out of 16 bits shows bit by bit each frequency and polarization.
"Pixel Data Quality 89" consist of one 1-byte integer and 4 out of 8 bits shows bit by bit each frequency and polarization.

```
764 C read array: pixel data quality for low
765       ! read
766       call H5Dopen_f(fhnd
767      +,'Pixel Data Quality 6 to 36',dhnd,ret)
768       sz1(1)=LMT
769       sz1(2)=AM2_DEF_SNUM_LO
770       call H5Dread_f(dhnd
771      +,H5T_NATIVE_INTEGER,i4d2hi,sz1,ret,H5S_ALL_F,H5S_ALL_F)
772       if(ret.lt.0)then
773         write(*,'(a,a)')'H5Dread error: '
774      +  ,'Pixel Data Quality 6 to 36'
775         call exit(1)
776       endif
777       call H5Dclose_f(dhnd,ret)
```

Read L1B pixel data quality

"Pixel Data Quality 6 to 36" are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to integer(1) type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value is 00.

```
778       ! cutoff overlap & separate
779       do j=1,num
780         do i=1,AM2_DEF_SNUM_LO
781          pdq06v(i,j)=0
782          if(btest(i4d2hi((i-1)*2+1,j+ovr),0))pdq06v(i,j)=1
783          if(btest(i4d2hi((i-1)*2+1,j+ovr),1))pdq06v(i,j)=10
784          if(btest(i4d2hi((i-1)*2+1,j+ovr),0)
785      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),1))pdq06v(i,j)=11
786          pdq06h(i,j)=0
787          if(btest(i4d2hi((i-1)*2+1,j+ovr),2))pdq06h(i,j)=1
788          if(btest(i4d2hi((i-1)*2+1,j+ovr),3))pdq06h(i,j)=10
789          if(btest(i4d2hi((i-1)*2+1,j+ovr),2)
790      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),3))pdq06h(i,j)=11
791          pdq07v(i,j)=0
792          if(btest(i4d2hi((i-1)*2+1,j+ovr),4))pdq07v(i,j)=1
793          if(btest(i4d2hi((i-1)*2+1,j+ovr),5))pdq07v(i,j)=10
794          if(btest(i4d2hi((i-1)*2+1,j+ovr),4)
795      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),5))pdq07v(i,j)=11
796          pdq07h(i,j)=0
797          if(btest(i4d2hi((i-1)*2+1,
798          if(btest(i4d2hi((i-1)*2+1,j+ovr),7))pdq07h(i,j)=10
799          if(btest(i4d2hi((i-1)*2+1,j+ovr),6)
800      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),7))pdq07h(i,j)=11
801         enddo
802       enddo
803       do j=num+1,LMT
804         pdq06h(:,j)=0
805         pdq06v(:,j)=0
806         pdq07h(:,j)=0
807         pdq07v(:,j)=0
808       enddo
```

Split the bit by bit information to each array of frequency and polarization.

Remove overlap scans also applied.

Read L1B land ocean flag * Numbers written on the left are row number of the sample program.

Data type of L1B land ocean flag is 1-byte integer and two dimensional (sample * (scan * channel)).
"Land_Ocean Flag 6 to 36" has 6 frequency channels (6, 7, 10, 18, 23, 36GHz).
"Land_Ocean Flag 89" has 2 frequency channels (89GHz-A and 89GHz-B).

For details to P.18.

```
848 C read array: land ocean flag for low
849       ! read
850       call H5Dopen_f(fhnd
851      +,'Land_Ocean Flag 6 to 36',dhnd,ret)
852       sz1(1)=LMT*6
853       sz1(2)=AM2_DEF_SNUM_LO
854       call H5Dread_f(dhnd
855      +,H5T_NATIVE_INTEGER,loflo,sz1,ret,H5S_ALL_F,H5S_ALL_F)
856       if(ret.lt.0)then
857         write(*,'(a,a)')'H5Dread error: '
858      +  ,'Land_Ocean Flag 6 to 36'
859         call exit(1)
860       endif
861       call H5Dclose_f(dhnd,ret)
```

Read L1B land ocean flag.

"Land_Ocean Flag 6 to 36" and "Land_Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to integer(1) type two dimensional array which is prepared for each frequency for convenience.

```
862       ! separate
863       do j=1,num+ovr*2
864         do i=1,AM2_DEF_SNUM_LO
865           lof06(i,j)=loflo(i,(num+ovr*2)*0+j)
866           lof07(i,j)=loflo(i,(num+ovr*2)*1+j)
867           lof10(i,j)=loflo(i,(num+ovr*2)*2+j)
868           lof18(i,j)=loflo(i,(num+ovr*2)*3+j)
869           lof23(i,j)=loflo(i,(num+ovr*2)*4+j)
870           lof36(i,j)=loflo(i,(num+ovr*2)*5+j)
871         enddo
872       enddo
873       ! cutoff overlap
874       do j=1,num
875         do i=1,AM2_DEF_SNUM_LO
876           lof06(i,j)=lof06(i,j+ovr)
877           lof07(i,j)=lof07(i,j+ovr)
878           lof10(i,j)=lof10(i,j+ovr)
879           lof18(i,j)=lof18(i,j+ovr)
880           lof23(i,j)=lof23(i,j+ovr)
881           lof36(i,j)=lof36(i,j+ovr)
882         enddo
883       enddo
884       do j=num+1,LMT
885         lof06(:,j)=0
886         lof07(:,j)=0
887         lof10(:,j)=0
888         lof18(:,j)=0
889         lof23(:,j)=0
890         lof36(:,j)=0
891       enddo
```

Split the data to each frequency.

Remove overlap scans also applied.

Read earth incidence * Numbers written on the left are row number of the sample program.

Earth incidence is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the earth incidence data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
934 C read array: earth incidence
935       ! read
936       call H5Dopen_f(fhnd
937      +,'Earth Incidence',dhnd,ret)
938       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)     ! get scale
939       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
940       call H5Aclose_f(ahnd,ret)                        ! get scale
941       sz1(1)=LMT
942       sz1(2)=AM2_DEF_SNUM_LO
943       call H5Dread_f(dhnd
944      +,H5T_NATIVE_REAL,ear_in,sz1,ret,H5S_ALL_F,H5S_ALL_F)
945       if(ret.lt.0)then
946         write(*,'(a,a)')'H5Dread error: '
947      +  ,'Earth Incidence'
948         call exit(1)
949       endif
950       call H5Dclose_f(dhnd,ret)
951       ! cutoff overlap & change scale
952       do j=1,num
953         do i=1,AM2_DEF_SNUM_LO
954           ear_in(i,j)=ear_in(i,j+ovr)
955           if(ear_in(i,j).gt.-32767)ear_in(i,j)=ear_in(i,j)*sca
956         enddo
957       enddo
958       do j=num+1,LMT
959         ear_in(:,j)=0
960       enddo
961       ! sample display
962       write(*,'(a,f9.2)')'ear_in(pixel=1,scan=1): ',ear_in(1,1)
```

Read scale factor.

Read earth incidence.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

Remove overlap scans.

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
993       call H5Fclose_f(fhnd,ret)
994       call H5close_f(ret)
995       end
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

8.1.2 Compile (Explanation of build_readL1B_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL1B_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11-12.
Since subroutines for time conversion and calculation of low frequency data are written in C language, specify C compiler,.
Intel compiler (ifort/icc) or PGI compiler (pgf90/pgcc) is required.

```
10 # compiler
11 fc=ifort
12 cc=icc
13
14 # source filename
15 fsrc="readL1B_hdf5.f"
16 csrc="amsr2time_.c amsr2latlon_.c"
17 obj=`echo $csrc|sed "s/¥.c/.o/g"`
18
19 # output filename
20 out=readL1B_hdf5_f
21
22 # library order
23 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
24
25 # c compile
26 cmd="$cc -g -c $csrc"
27 echo $cmd
28 $cmd
29
30 # f compile
31 cmd="$fc -g $fsrc $obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
32 echo $cmd
33 $cmd
34
35 # garbage
36 rm -f *.o
```

The execution example of “build_readL1B_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1B_hdf5_f.sh
icc -g -c amsr2time_.c amsr2latlon_.c
ifort -g readL1B_hdf5.f amsr2time_.o amsr2latlon_.o -o readL1B_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.1.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1B_hdf5_f” is shown as follows.

```
$ ./readL1B_hdf5_f GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGBTBR_0000000.h5
NumberOfScans(RETRIEVE BY ARRAY SIZE):        1979
OverlapScans(FIXED VALUE):          20
limit of NumberOfScans =   2200
CoRegistrationParameterA1(FIXED VALUE): 6G- 1.25000,7G- 1.00000,10G- 1.25000,18G- 1.25000,23G- 1.25000,36G- 1.00000
CoRegistrationParameterA2(FIXED VALUE): 6G- 0.00000,7G--0.10000,10G--0.25000,18G- 0.00000,23G--0.25000,36G- 0.00000
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leapsec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time(scan=1): 2012/07/26 11:45:43
latlon89a(pixel=1,scan=1): ( -73.3289, 136.7714)
latlon89b(pixel=1,scan=1): ( -73.4038, 137.1498)
latlonlm(pixel=1,scan=1): ( -73.3538, 136.6228)
latlon06(pixel=1,scan=1): ( -73.3592, 136.6213)
latlon07(pixel=1,scan=1): ( -73.3497, 136.6429)
latlon10(pixel=1,scan=1): ( -73.3506, 136.6001)
latlon18(pixel=1,scan=1): ( -73.3592, 136.6213)
latlon23(pixel=1,scan=1): ( -73.3506, 136.6001)
latlon36(pixel=1,scan=1): ( -73.3532, 136.6514)
tb06h(pixel=1,scan=1):    173.28
tb06v(pixel=1,scan=1):    208.22
tb07h(pixel=1,scan=1):    173.07
tb07v(pixel=1,scan=1):    207.54
tb10h(pixel=1,scan=1):    170.94
tb10v(pixel=1,scan=1):    204.95
tb18h(pixel=1,scan=1):    164.85
tb18v(pixel=1,scan=1):    199.84
tb23h(pixel=1,scan=1):    163.22
tb23v(pixel=1,scan=1):    196.53
tb36h(pixel=1,scan=1):    156.56
tb36v(pixel=1,scan=1):    186.39
tb89ah(pixel=1,scan=1):    163.76
tb89av(pixel=1,scan=1):    179.27
tb89bh(pixel=1,scan=1):    170.60
tb89bv(pixel=1,scan=1):    188.16
pdq06h(pixel=1,scan=1):   0
pdq06v(pixel=1,scan=1):   0
pdq07h(pixel=1,scan=1):   0
pdq07v(pixel=1,scan=1):   0
pdq10h(pixel=1,scan=1):   0
pdq10v(pixel=1,scan=1):   0
pdq18h(pixel=1,scan=1):   0
pdq18v(pixel=1,scan=1):   0
pdq23h(pixel=1,scan=1):   0
pdq23v(pixel=1,scan=1):   0
pdq36h(pixel=1,scan=1):   0
pdq36v(pixel=1,scan=1):   0
pdq89ah(pixel=1,scan=1):   0
pdq89av(pixel=1,scan=1):   0
pdq89bh(pixel=1,scan=1):   0
pdq89bv(pixel=1,scan=1):   0
lof06(pixel=1,scan=1): 100
lof07(pixel=1,scan=1): 100
lof10(pixel=1,scan=1): 100
lof18(pixel=1,scan=1): 100
lof23(pixel=1,scan=1): 100
lof36(pixel=1,scan=1): 100
lof89a(pixel=1,scan=1): 100
lof89b(pixel=1,scan=1): 100
ear_in(pixel=1,scan=1):    55.20
ear_az(pixel=1,scan=1):   144.76
```

8.2 Read L1R Product

8.2.1 FORTRAN90 sample (readL1R_hdf5.f) program

Sample program (readL1R_hdf5.f) which reads the metadata and the datasets of L1R is shown below. Latitude and longitude of low frequency data are extracted from observation point of 89GHz-A. Values of data are dumped to the standard output.

Only part of L1R brightness temperature data are handled in this sample program. Please refer to "3.8 Level1 Resampling Products (L1R)" on page 18.

Subroutines are prepared for the conversion of TAI93.

Metadata	Datasets
* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. * NumberOfScans are acquired from the array dimension of the dataset. Fixed values are used for the variable below. * OverlapScans	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B - Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Brightness Temperature (res06,6.9GHz,H) - Brightness Temperature (res06,6.9GHz,V) - Brightness Temperature (res06,7.3GHz,H) - Brightness Temperature (res06,7.3GHz,V) - Brightness Temperature (res10,10.7GHz,H) - Brightness Temperature (res10,10.7GHz,V) - Brightness Temperature (res23,18.7GHz,H) - Brightness Temperature (res23,18.7GHz,V) - Brightness Temperature (res23,23.8GHz,H) - Brightness Temperature (res23,23.8GHz,V) - Brightness Temperature (res36,36.5GHz,H) - Brightness Temperature (res36,36.5GHz,V) - Brightness Temperature (res36,89.0GHz,H) - Brightness Temperature (res36,89.0GHz,V) - Brightness Temperature (original,89GHz-A,H) - Brightness Temperature (original,89GHz-A,V) - Brightness Temperature (original,89GHz-B,H) - Brightness Temperature (original,89GHz-B,V) * Pixel Data Quality 6 to 36 - Pixel Data Quality 89 * Land_Ocean Flag 6 to 36 - Land_Ocean Flag 89 * Earth Incidence - Earth Azimuth
Data extracted from observation point of 89GHz-A	
* Latitude and longitude of low frequency data For details to P.20	

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 4 C include
 5      include 'amsr2time_f.h'
 :
 8 C fixed value
 9      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
10      integer(4),parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
11      integer(4),parameter::AM2_DEF_SNUM_LO=243 ! low  resolution pixel width
12 C interface variable
13      integer(4) i,j          ! loop variable
14      integer(4) ret          ! return status
15      character(len=512) buf  ! text buffer
16      character(len=512) fn   ! filename
17      integer(HID_T) fhnd     ! file handle
18      integer(HID_T) ahnd     ! attribute handle
19      integer(HID_T) atyp     ! attribute type
20      integer(HID_T) dhnd     ! dataset handle
21      integer(HID_T) dtyp     ! dataset type
22      integer(HID_T) shnd     ! dataspace handle
23      integer(HSIZE_T) sz1(3) ! array size 1
24      integer(HSIZE_T) sz2(3) ! array size 2
 :
35      integer(4) num          ! NumberOfScans
36      integer(4) ovr          ! OverlapScans
 :
39      parameter(ovr=20)
 :
45      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
46      real(4) lat89ar(AM2_DEF_SNUM_HI,LMT) ! lat for 89a altitude revised
47      real(4) lat89br(AM2_DEF_SNUM_HI,LMT) ! lat for 89b altitude revised
 :
49      real(4) lon89ar(AM2_DEF_SNUM_HI,LMT) ! lon for 89a altitude revised
50      real(4) lon89br(AM2_DEF_SNUM_HI,LMT) ! lon for 89b altitude revised
 :
52      real(4) tb06h06(AM2_DEF_SNUM_LO,LMT) ! tb for 06h, resolution 06G
53      real(4) tb06v06(AM2_DEF_SNUM_LO,LMT) ! tb for 06v, resolution 06G
 :
70      integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT)  ! pixel data quality for 06h
71      integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT)  ! pixel data quality for 06v
 :
79      integer(1) lof06(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 06
80      integer(1) lof10(AM2_DEF_SNUM_LO,LMT)   ! land ocean flag for 10
 :
86      real(4) ear_in(AM2_DEF_SNUM_LO,LMT) ! earth incidence
87      real(4) ear_az(AM2_DEF_SNUM_LO,LMT) ! earth azimuth
```

Employ the USE statement to make use of the HDF5 functions in the program.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.
※Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. Fixed values are set in the program.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
 96 C hdf5 initialize
 97       call H5open_f(ret)
 98       if(ret.lt.0)then
 99         write(*,'(a,i12)')'h5open_f error: ',ret
100         call exit(1)
101       endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allow read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
102 C open
103       call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
104       if(ret.lt.0)then
105         write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
106         call exit(1)
107       endif
```

Specify NumberOfScans * Numbers written on the left are row number of the sample program.

Since the metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library, you need to acquire “NumberOfScans” from array dimension of the dataset “Scan Time”. First, open the dataset by using “H5Dopen_f” function. Next, get the dataspace type by using “H5Dget_space_f” function. Then you'll be able to acquire the array dimension of the dataset by using “H5Sget_simple_extent_dims_f” function.Close the dataset and dataspace identifier after you read the data.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension.
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
108 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
109 C calculate NumberOfScans from array size and OverlapScans
110       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
111       call H5Dget_space_f(dhnd,shnd,ret)
112       call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
113       if(ret.lt.0)then
114         write(*,'(a)')'H5Sget_simple_extent_dims error: Scan Time'
115         call exit(1)
116       endif
117       call H5Sclose_f(shnd,ret)
118       call H5Dclose_f(dhnd,ret)
119       num=sz1(1)-ovr*2
120       write(*,'(a,i12)')'NumberOfScans(RETRIEVE BY ARRAY SIZE): ',num
121       write(*,'(a,i12)')'OverlapScans(FIXED VALUE): ',ovr
```

NumberOfScans is represented as below.
Sum of scans in product - (OverlapScans x 2)

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen_f function.
Dimension size of the dataset are required when reading.
Overlap scans contained at both end of L1 data should be removed.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

For details to P.13.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
153 C read array: scantime
154       ! read
155       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
156       sz1(1)=LMT
157       call H5Dread_f(dhnd
158      +,H5T_NATIVE_DOUBLE,r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
159       if(ret.lt.0)then
160         write(*,'(a,a)')'H5Dread error: ','Scan Time'
161         call exit(1)
162       endif
163       call H5Dclose_f(dhnd,ret)
164       ! cutoff overlap
165       do j=1,num
166         r8d1(j)=r8d1(j+ovr)
167       enddo
168       do j=num+1,LMT
169         r8d1(j)=0
170       enddo
171       ! convert
172       call amsr2time(num,r8d1,st)
173       ! sample display
174       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
175      +'time(scan=1): '
176      +,st(1)%year
177      +,st(1)%month
178      +,st(1)%day
179      +,st(1)%hour
180      +,st(1)%minute
181      +,st(1)%second
```

Read scanning time.

Remove overlap scans.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
call amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude of 89GHz.
Overlap scans contained at both end of L1 data should be removed.
For the latitude and longitude of low frequency data, extract odd sample number of 89GHz-A observation points (origin 1).

For details to P.20.

```
182 C read array: latlon for 89a altitude revised
183       ! read lat
184       call H5Dopen_f(fhnd
185      +,'Latitude of Observation Point for 89A',dhnd,ret)
186       sz1(1)=LMT
187       sz1(2)=AM2_DEF_SNUM_HI
188       call H5Dread_f(dhnd
189      +,H5T_NATIVE_REAL,lat89ar,sz1,ret,H5S_ALL_F,H5S_ALL_F)
190       if(ret.lt.0)then
191         write(*,'(a,a)')'H5Dread error: '
192      +  ,'Latitude of Observation Point for 89A'
193         call exit(1)
194       endif
195       call H5Dclose_f(dhnd,ret)
196       ! read lon
197       call H5Dopen_f(fhnd
198      +,'Longitude of Observation Point for 89A',dhnd,ret)
199       sz1(1)=LMT
200       sz1(2)=AM2_DEF_SNUM_HI
201       call H5Dread_f(dhnd
202      +,H5T_NATIVE_REAL,lon89ar,sz1,ret,H5S_ALL_F,H5S_ALL_F)
203       if(ret.lt.0)then
204         write(*,'(a,a)')'H5Dread error: '
205      +  ,'Longitude of Observation Point for 89A'
206         call exit(1)
207       endif
208       call H5Dclose_f(dhnd,ret)
209       ! cutoff overlap
210       do j=1,num
211         lat89ar(:,j)=lat89ar(:,j+ovr)
212         lon89ar(:,j)=lon89ar(:,j+ovr)
213       enddo
214       do j=num+1,LMT
215         lat89ar(:,j)=0
216         lon89ar(:,j)=0
217       enddo
 :

260 C read array: latlon for low resolution
261       do j=1,num
262         do i=1,AM2_DEF_SNUM_LO
263           latlr(i,j)=lat89ar(i*2-1,j)
264           lonlr(i,j)=lon89ar(i*2-1,j)
265         enddo
266       enddo
```

Read latitude.

Read longitude.

Remove overlap scans.

Extract the position of low frequency data from the observation point of 89GHz-A.

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer (0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value (65534) must be excluded.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

```
269 C read array: tb for 06h, resolution 06G
270       ! read
271       call H5Dopen_f(fhnd
272      +,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
273       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
274       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
275       call H5Aclose_f(ahnd,ret)                    ! get scale
276       sz1(1)=LMT
277       sz1(2)=AM2_DEF_SNUM_LO
278       call H5Dread_f(dhnd
279      +,H5T_NATIVE_REAL,tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
280       if(ret.lt.0)then
281         write(*,'(a,a)')'H5Dread error: '
282      +  ,'Brightness Temperature (res06,6.9GHz,H)'
283         call exit(1)
284       endif
285       call H5Dclose_f(dhnd,ret)
286       ! cutoff overlap & convert to unsignd & change scale
287       do j=1,num
288         do i=1,AM2_DEF_SNUM_LO
289           tb06h06(i,j)=tb06h06(i,j+ovr)
290           if(tb06h06(i,j).lt.65534)tb06h06(i,j)=tb06h06(i,j)*sca
291         enddo
292       enddo
293       do j=num+1,LMT
294         tb06h06(:,j)=0
295       enddo
296       ! sample display
297       write(*,'(a,f9.2)')'tb06h06(pixel=1,scan=1): ',tb06h06(1,1)
```

Read scale factor.

Read brightness temperature.

Exclude the missing value (65535) and error value (65534) when scale handling is applied.

Remove overlap scans.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Scale handling is automatically applied when using AMTK. For details to P.22.

Read L1R pixel data quality * Numbers written on the left are row number of the sample program.

"Pixel Data Quality 6 to 36" consist of two 1-byte integer and 12 out of 16 bits shows bit by bit each frequency and polarization.
"Pixel Data Quality 89" consist of one 1-byte integer and 4 out of 8 bits shows bit by bit each frequency and polarization.

```
791 C read array: pixel data quality for low
792       ! read
793       call H5Dopen_f(fhnd
794      +,'Pixel Data Quality 6 to 36',dhnd,ret)
795       sz1(1)=LMT
796       sz1(2)=AM2_DEF_SNUM_LO
797       call H5Dread_f(dhnd
798      +,H5T_NATIVE_INTEGER,i4d2hi,sz1,ret,H5S_ALL_F,H5S_ALL_F)
799       if(ret.lt.0)then
800         write(*,'(a,a)')'H5Dread error: '
801      +  ,'Pixel Data Quality 6 to 36'
802         call exit(1)
803       endif
804       call H5Dclose_f(dhnd,ret)
```

Read L1R pixel data quality

"Pixel Data Quality 6 to 36" are acquired as one data, respectively.
Since the stored value is 2bit, we split the data to integer(1) type two dimensional array which is prepared for each frequency and polarization for convenience.
Information for RFI (Radio Frequency Interference) is stored in pixel data quality.
If pixel has affected by RFI, the value of pixel data quality is set as 11, has possibly affected by RFI, the value is 10, and otherwise the value 00.

```
805       ! cutoff overlap & separate
806       do j=1,num
807         do i=1,AM2_DEF_SNUM_LO
808          pdq06v(i,j)=0
809          if(btest(i4d2hi((i-1)*2+1,j+ovr),0))pdq06v(i,j)=1
810          if(btest(i4d2hi((i-1)*2+1,j+ovr),1))pdq06v(i,j)=10
811          if(btest(i4d2hi((i-1)*2+1,j+ovr),0)
812      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),1))pdq06v(i,j)=11
813          pdq06h(i,j)=0
814          if(btest(i4d2hi((i-1)*2+1,j+ovr),2))pdq06h(i,j)=1
815          if(btest(i4d2hi((i-1)*2+1,j+ovr),3))pdq06h(i,j)=10
816          if(btest(i4d2hi((i-1)*2+1,j+ovr),2)
817      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),3))pdq06h(i,j)=11
818          pdq07v(i,j)=0
819          if(btest(i4d2hi((i-1)*2+1,j+ovr),4))pdq07v(i,j)=1
820          if(btest(i4d2hi((i-1)*2+1,j+ovr),5))pdq07v(i,j)=10
821          if(btest(i4d2hi((i-1)*2+1,j+ovr),4)
822      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),5))pdq07v(i,j)=11
823          pdq07h(i,j)=0
824          if(btest(i4d2hi(
825          if(btest(i4d2hi(
826          if(btest(i4d2hi((i-1)*2+1,j+ovr),6)
827      +     .and.btest(i4d2hi((i-1)*2+1,j+ovr),7))pdq07h(i,j)=11
828         enddo
829       enddo
830       do j=num+1,LMT
831         pdq06h(:,j)=0
832         pdq06v(:,j)=0
833         pdq07h(:,j)=0
834         pdq07v(:,j)=0
835       enddo
```

Split the bit by bit information to each array of frequency and polarization.

Remove overlap scans also applied.

Read L1R land ocean flag * Numbers written on the left are row number of the sample program.

Data type of L1R land ocean flag is 1-byte integer and two dimensional (sample * (scan * channel)).
"Land_Ocean Flag 6 to 36" has 6 frequency channels (6, 7, 10, 18, 23, 36GHz).
"Land_Ocean Flag 89" has 2 frequency channels (89GHz-A and 89GHz-B).

For details to P.18.

```
875 C read array: land ocean flag for low
876       ! read
877       call H5Dopen_f(fhnd
878      +,'Land_Ocean Flag 6 to 36',dhnd,ret)
879       sz1(1)=LMT*6
880       sz1(2)=AM2_DEF_SNUM_LO
881       call H5Dread_f(dhnd
882      +,H5T_NATIVE_INTEGER,loflo,sz1,ret,H5S_ALL_F,H5S_ALL_F)
883       if(ret.lt.0)then
884         write(*,'(a,a)')'H5Dread error:
885      +  ,'Land_Ocean Flag 6 to 36'
886         call exit(1)
887       endif
888       call H5Dclose_f(dhnd,ret)
```

Read L1R land ocean flag.

"Land_Ocean Flag 6 to 36" and "Land_Ocean Flag 89" is acquired as one data, respectively.
Since the stored value is 0 to 100, we split the data to integer(1) type two dimensional array which is prepared for each frequency for convenience.

```
889       ! separate
890       do j=1,num+ovr*2
891         do i=1,AM2_DEF_SNUM_LO
892           lof06(i,j)=loflo(i,(num+ovr*2)*0+j)
893           lof10(i,j)=loflo(i,(num+ovr*2)*1+j)
894           lof23(i,j)=loflo(i,(num+ovr*2)*2+j)
895           lof36(i,j)=loflo(i,(num+ovr*2)*3+j)
896         enddo
897       enddo
898       ! cutoff overlap
899       do j=1,num
900         do i=1,AM2_DEF_SNUM_LO
901           lof06(i,j)=lof06(i,j+ovr)
902           lof10(i,j)=lof10(i,j+ovr)
903           lof23(i,j)=lof23(i,j+ovr)
904           lof36(i,j)=lof36(i,j+ovr)
905         enddo
906       enddo
907       do j=num+1,LMT
908         lof06(:,j)=0
909         lof10(:,j)=0
910         lof23(:,j)=0
911         lof36(:,j)=0
912       enddo
```

Split the data to each frequency.

Remove overlap scans also applied.

Read earth incidence * Numbers written on the left are row number of the sample program.

Earth incidence is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the earth incidence data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

```
953 C read array: earth incidence
954       ! read
955       call H5Dopen_f(fhnd
956      +,'Earth Incidence',dhnd,ret)
957       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)     ! get scale
958       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
959       call H5Aclose_f(ahnd,ret)                        ! get scale
960       sz1(1)=LMT
961       sz1(2)=AM2_DEF_SNUM_LO
962       call H5Dread_f(dhnd
963      +,H5T_NATIVE_REAL,ear_in,sz1,ret,H5S_ALL_F,H5S_ALL_F)
964       if(ret.lt.0)then
965         write(*,'(a,a)')'H5Dread error: '
966      +  ,'Earth Incidence'
967         call exit(1)
968       endif
969       call H5Dclose_f(dhnd,ret)
970       ! cutoff overlap & change scale
971       do j=1,num
972         do i=1,AM2_DEF_SNUM_LO
973           ear_in(i,j)=ear_in(i,j+ovr)
974           if(ear_in(i,j).gt.-32767)ear_in(i,j)=ear_in(i,j)*sca
975         enddo
976       enddo
977       do j=num+1,LMT
978         ear_in(:,j)=0
979       enddo
980       ! sample displa
981       write(*,'(a,f9.2)')'ear_in(pixel=1,scan=1): ',ear_in(1,1)
```

Read scale factor.

Read earth incidence.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

Remove overlap scans.

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
1012      call H5Fclose_f(fhnd,ret)
1013      call H5close_f(ret)
1014      end
```

Terminate access to an HDF5 file.
ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
ret = H5close();
ret: [Return value] Error: A negative value is returned.

8.2.2 Compile (Explanation of build_readL1R_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL1R_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11-12.
Since subroutine for time conversion is written in C language, specify C compiler.
Intel compiler (ifort/icc) or PGI compiler (pgf90/pgcc) is required.

```
10 # compiler
11 fc=ifort
12 cc=icc
13
14 # source filename
15 fsrc="readL1R_hdf5.f"
16 csrc="amsr2time_.c "
17 obj=`echo $csrc|sed "s/¥.c/.o/g"`
18
19 # output filename
20 out=readL1R_hdf5_f
21
22 # library order
23 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
24
25 # c compile
26 cmd="$cc -g -c $csrc"
27 echo $cmd
28 $cmd
29
30 # f compile
31 cmd="$fc -g $fsrc $obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
32 echo $cmd
33 $cmd
34
35 # garbage
36 rm -f *.o
```

The execution example of “build_readL1R_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL1R_hdf5_f.sh
icc -g -c amsr2time_.c amsr2latlon_.c
ifort -g readL1R_hdf5.f amsr2time_.o amsr2latlon_.o -o readL1R_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.2.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL1R_hdf5_f” is shown as follows.

```
$ ./readL1R_hdf5_f GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGRTBR_0000000.h5
NumberOfScans(RETRIEVE BY ARRAY SIZE):        1979
OverlapScans(FIXED VALUE):          20
limit of NumberOfScans =   2200
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leapsec.dat
amsr2time: year=1993 month= 7 tai93sec=   156384001.00
amsr2time: year=1994 month= 7 tai93sec=   471744002.00
amsr2time: year=1996 month= 1 tai93sec=   946080003.00
amsr2time: year=1997 month= 7 tai93sec=  1418688004.00
amsr2time: year=1999 month= 1 tai93sec=  1893024005.00
amsr2time: year=2006 month= 1 tai93sec=  4102272006.00
amsr2time: year=2009 month= 1 tai93sec=  5049216007.00
amsr2time: year=2012 month= 7 tai93sec=  6152544008.00
amsr2time: number of leap second = 8
time(scan=1): 2012/07/26 11:45:43
latlon89ar(pixel=1,scan=1): ( -73.3581, 136.8432)
latlon89br(pixel=1,scan=1): ( -73.4328, 137.2216)
latlonlr(pixel=1,scan=1): ( -73.3581, 136.8432)
tb06h06(pixel=1,scan=1):    173.41
tb06v06(pixel=1,scan=1):    208.09
tb07h06(pixel=1,scan=1):    173.07
tb07v06(pixel=1,scan=1):    207.11
tb10h10(pixel=1,scan=1):    170.40
tb10v10(pixel=1,scan=1):    204.58
tb18h23(pixel=1,scan=1):    165.83
tb18v23(pixel=1,scan=1):    199.41
tb23h23(pixel=1,scan=1):    163.55
tb23v23(pixel=1,scan=1):    195.90
tb36h36(pixel=1,scan=1):    153.55
tb36v36(pixel=1,scan=1):    183.95
tb89h36(pixel=1,scan=1):    163.86
tb89v36(pixel=1,scan=1):    181.05
tb89ah(pixel=1,scan=1):    163.76
tb89av(pixel=1,scan=1):    179.27
tb89bh(pixel=1,scan=1):    170.60
tb89bv(pixel=1,scan=1):    188.16
pdq06h(pixel=1,scan=1):   0
pdq06v(pixel=1,scan=1):   0
pdq07h(pixel=1,scan=1):   0
pdq07v(pixel=1,scan=1):   0
pdq10h(pixel=1,scan=1):   0
pdq10v(pixel=1,scan=1):   0
pdq18h(pixel=1,scan=1):   0
pdq18v(pixel=1,scan=1):   0
pdq23h(pixel=1,scan=1):   0
pdq23v(pixel=1,scan=1):   0
pdq36h(pixel=1,scan=1):   0
pdq36v(pixel=1,scan=1):   0
pdq89ah(pixel=1,scan=1):   0
pdq89av(pixel=1,scan=1):   0
pdq89bh(pixel=1,scan=1):   0
pdq89bv(pixel=1,scan=1):   0
lof06(pixel=1,scan=1): 100
lof10(pixel=1,scan=1): 100
lof23(pixel=1,scan=1): 100
lof36(pixel=1,scan=1): 100
lof89a(pixel=1,scan=1): 100
lof89b(pixel=1,scan=1): 100
ear_in(pixel=1,scan=1):    55.20
ear_az(pixel=1,scan=1):   144.76
```

8.3 Read L2 Low Resolution Product

L2 Low Resolution Products are Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW). For Precipitation(PRC) product, please refer to “8.4 Read L2 High Resolution Product” on page 228.

8.3.1 Fortran sample program (readL2L_hdf5.f)

Sample program (readL2L_hdf5.f) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93.

Metadata	Datasets
* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. * NumberOfScans are acquired from the array dimension of the dataset. Fixed values are used for the variable below. * OverlapScans	* Scan Time * Latitude of Observation Point * Longitude of Observation Point * Geophysical Data * Pixel Data Quality For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 4 C include
 5      include 'amsr2time_f.h'
 :
 8 C fixed value
 9      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
10      integer(4),parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
11      integer(4),parameter::AM2_DEF_SNUM_LO=243 ! low  resolution pixel width
 :
15 C interface variable
16      integer(4) i,j          ! loop variable
17      integer(4) ret          ! return status
18      character(len=512) buf  ! text buffer
19      character(len=512) fn   ! filename
20      integer(HID_T) fhnd     ! file handle
21      integer(HID_T) ahnd     ! attribute handle
22      integer(HID_T) atyp     ! attribute type
23      integer(HID_T) dhnd     ! dataset handle
24      integer(HID_T) shnd     ! dataspace handle
25      integer(HSIZE_T) sz1(3) ! array size 1
26      integer(HSIZE_T) sz2(3) ! array size 2
 :
37      integer(4) num          ! NumberOfScans
38      integer(4) ovr          ! OverlapScans
39      parameter(ovr=0)

43 C array data
44      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
45      real(4) lat(AM2_DEF_SNUM_LO,LMT)  ! lat
46      real(4) lon(AM2_DEF_SNUM_LO,LMT)  ! lon
47      real(4) geo1(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 1
48      real(4) geo2(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 2
49      real(4) geotmp(AM2_DEF_SNUM_LO,LMT*2) ! geophysical data temporary

50      integer(4) pdq1(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 1
51      integer(4) pdq2(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 2
52      integer(4) pdqtmp(AM2_DEF_SNUM_LO,LMT*2) ! pixel data quality temporary
```

Employ the USE statement to make use of the HDF5 functions in the program.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.
※Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. Fixed values are set in the program.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
61 C hdf5 initialize
62       call H5open_f(ret)
63       if(ret.lt.0)then
64         write(*,'(a,i12)')'h5open_f error: ',ret
65         call exit(1)
66       endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allow read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
67 C open
68       call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
69       if(ret.lt.0)then
70         write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
71         call exit(1)
72       endif
```

Specify NumberOfScans * Numbers written on the left are row number of the sample program.

Since the metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library, you need to acquire “NumberOfScans” from array dimension of the dataset “Scan Time”. First, open the dataset by using “H5Dopen_f” function. Next, get the dataspace type by using “H5Dget_space_f” function. Then you'll be able to acquire the array dimension of the dataset by using “H5Sget_simple_extent_dims_f” function.Close the dataset and dataspace identifier after you read the data.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
 90 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
 91 C calculate NumberOfScans from array size and OverlapScans
 92       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
 93       call H5Dget_space_f(dhnd,shnd,ret)
 94       call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
 95       if(ret.lt.0)then
 96         write(*,'(a)')'H5Sget_simple_extent_dims error: Scan Time'
 97         call exit(1)
 98       endif
 99       call H5Sclose_f(shnd,ret)
100       call H5Dclose_f(dhnd,ret)
101       num=sz1(1)-ovr*2
102       write(*,'(a,i12)')'NumberOfScans(RETRIEVE BY ARRAY SIZE): ',num
103       write(*,'(a,i12)')'OverlapScans(FIXED VALUE): ',ovr
```

NumberOfScans is represented as below.
Sum of scans in product - (OverlapScans x 2)

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen function.
Dimension size of the dataset are required when reading.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
111 C read array: scantime
112       ! read
          call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
          sz1(1)=LMT
          call H5Dread_f(dhnd
         +,H5T_NATIVE_DOUBLE,r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
          if(ret.lt.0)then
            write(*,'(a,a)')'H5Dread error: ','Scan Time'
            call exit(1)
          endif
          call H5Dclose_f(dhnd,ret)
          ! convert
          call amsr2time(num,r8d1,st)
          ! sample display
          write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
         +'time(scan=1): '
         +,st(1)%year
         +,st(1)%month
         +,st(1)%day
         +,st(1)%hour
         +,st(1)%minute
         +,st(1)%second
```

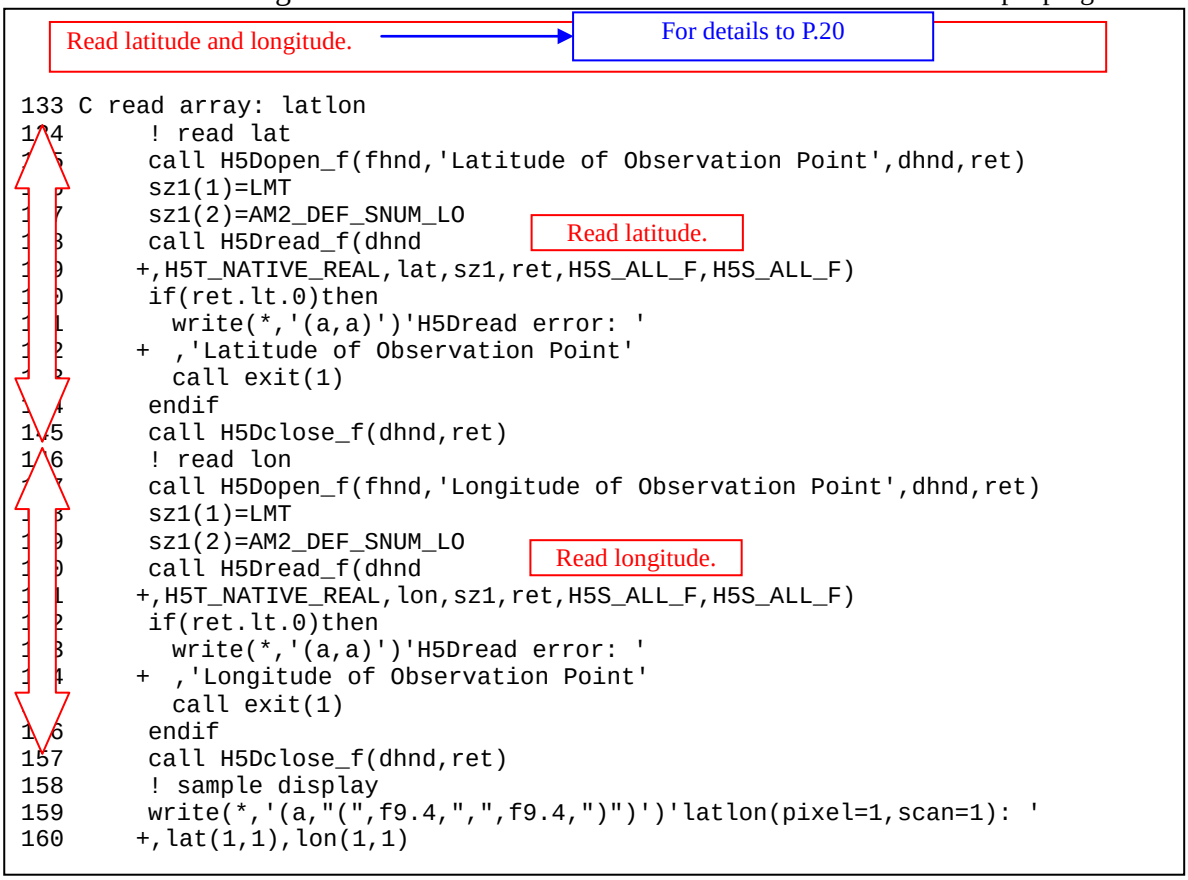
Read scanning time.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.




```
133 C read array: latlon
134       ! read lat
135       call H5Dopen_f(fhnd,'Latitude of Observation Point',dhnd,ret)
136       sz1(1)=LMT
137       sz1(2)=AM2_DEF_SNUM_LO
138       call H5Dread_f(dhnd
139      +,H5T_NATIVE_REAL,lat,sz1,ret,H5S_ALL_F,H5S_ALL_F)
140       if(ret.lt.0)then
141         write(*,'(a,a)')'H5Dread error: '
142      +  ,'Latitude of Observation Point'
143         call exit(1)
144       endif
145       call H5Dclose_f(dhnd,ret)
146       ! read lon
147       call H5Dopen_f(fhnd,'Longitude of Observation Point',dhnd,ret)
148       sz1(1)=LMT
149       sz1(2)=AM2_DEF_SNUM_LO
150       call H5Dread_f(dhnd
151      +,H5T_NATIVE_REAL,lon,sz1,ret,H5S_ALL_F,H5S_ALL_F)
152       if(ret.lt.0)then
153         write(*,'(a,a)')'H5Dread error: '
154      +  ,'Longitude of Observation Point'
155         call exit(1)
156       endif
157       call H5Dclose_f(dhnd,ret)
158       ! sample display
159       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon(pixel=1,scan=1): '
160      +,lat(1,1),lon(1,1)
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

Open an attribute.
call H5Aopen_f(dhnd,nam,ahnd,ret)
dhnd: Object id.
nam: Name of attribute.
ahnd: [Return value] Attribute id
ret: [Return value] Error: A negative value is returned.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

```
161 C read array: geophysical data for 1 layer
162       if((gid(30:32).ne.'SND').and.(gid(30:32).ne.'SST'))then
163         call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
164         ! get scale
165         call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
166         call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,
167         call H5Aclose_f(ahnd,ret)
1 8         ! read
            sz1(1)=LMT
1 0         sz1(2)=AM2_DEF_SNUM_LO
1 1         call H5Dread_f(dhnd
1 2      +  ,H5T_NATIVE_REAL,geo1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
1 3         if(ret.lt.0)then
1 4           write(*,'(a,a)')'H5Dread error: '
1 5      +    ,'Geophysical Data'
1 6           call exit(1)
            endif
1 8         call H5Dclose_f(dhnd,ret)
179         ! change scale
1 0         do j=1,num
1 1           do i=1,AM2_DEF_SNUM_LO
1 2             if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
1 3           enddo
1 4         enddo
1           endif
```

Read scale factor.

Read geophysical quantity.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
C read array: geophysical data for 2 layer
     if((gid(30:32).eq.'SND').or.(gid(30:32).eq.'SST'))then
       call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
       ! get scale
       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
       call H5Aclose_f(ahnd,ret)
       ! read
       sz1(1)=LMT*2
       sz1(2)=AM2_DEF_SNUM_LO
       call H5Dread_f(dhnd
    +  ,H5T_NATIVE_REAL,geotmp,sz1,ret,H5S_ALL_F,H5S_ALL_F)
       if(ret.lt.0)then
         write(*,'(a,a)')'H5Dread error: '
    +   ,'Geophysical Data'
         call exit(1)
       endif
       call H5Dclose_f(dhnd,ret)
       ! separate & change scale
       do j=1,num
         do i=1,AM2_DEF_SNUM_LO
           geo1(i,j)=geotmp((i-1)*2+1,(j-1)*2+1)
           geo2(i,j)=geotmp((i-1)*2+2,(j-1)*2+1)
           if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
           if(geo2(i,j).gt.-32767)geo2(i,j)=geo2(i,j)*sca
         enddo
       enddo
     endif
```

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.
Snow depth product and Sea surface temperature product have two layer of pixel data quality.

```
214 C read array: pixel data quality for 1 layer
215       if((gid(30:32).ne.'SND').and.(gid(30:32).ne.'SST'))then
216         call H5Dopen_f(fhnd,'Pixel Data Quality',dhnd,ret)
217         sz1(1)=LMT
218         sz1(2)=AM2_DEF_SNUM_LO
219         call H5Dread_f(dhnd
220      +  ,H5T_NATIVE_INTEGER,pdq1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
221         if(ret.lt.0)then
222           write(*,'(a,a)')'H5Dread error: '
223      +    ,'Pixel Data Quality'
224           call exit(1)
225         endif
226         call H5Dclose_f(dhnd,ret)
227       endif
```

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
228 C read array: pixel data quality for 2 layer
229       if((gid(30:32).eq.'SND').or.(gid(30:32).eq.'SST'))then
230         ! read
231         call H5Dopen_f(fhnd,'Pixel Data Quality',dhnd,ret)
232         sz1(1)=LMT*2
233         sz1(2)=AM2_DEF_SNUM_LO
234         call H5Dread_f(dhnd
235      +  ,H5T_NATIVE_INTEGER,pdqtmp,sz1,ret,H5S_ALL_F,H5S_ALL_F)
236         if(ret.lt.0)then
237           write(*,'(a,a)')'H5Dread error: '
238      +    ,'Pixel Data Quality'
239           call exit(1)
240         endif
241         call H5Dclose_f(dhnd,ret)
242         ! separate
243         do j=1,num
244           do i=1,AM2_DEF_SNUM_LO
245             pdq1(i,j)=pdqtmp(i,num*0+j)
246             pdq2(i,j)=pdqtmp(i,num*1+j)
247           enddo
248         enddo
249       endif
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
1027 C close
1028       call H5Fclose_f(fhnd,ret)
1029       call H5close_f(ret)
```

Terminate access to an HDF5 file.
call H5Fclose_f(fhnd,ret)
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
call H5close_f(ret)
ret: [Return value] Error: A negative value is returned.

8.3.2 Compile (Explanation of build_readL2L_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL2L_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
  1 #!/bin/sh
  2
  3 ### environment
  4 export LANG=C
  5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
  6 # library directory
  7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
  8 SZIP=/home/user1/util/szip_2.1
  9
```

Specify the compiler you use in row number 11-12.
Since subroutine for time conversion is written in C language, specify C compiler.
Intel compiler (ifort/icc) or PGI compiler (pgf90/pgcc) is required.

```
 10 # compiler
 11 fc=ifort
 12 cc=icc
 13
 14 # source filename
 15 fsrc="readL2L_hdf5.f"
 16 csrc="amsr2time_.c"
 17 obj=`echo $csrc|sed "s/¥.c/.o/g"`
 18
 19 # output filename
 20 out=readL2L_hdf5_f
 21
 22 # library order
 23 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
 24
 25 # c compile
 26 cmd="$cc -g -c $csrc"
 27 echo $cmd
 28 $cmd
 29
 30 # f compile
 31 cmd="$fc -g $fsrc $obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
 32 echo $cmd
 33 $cmd
 34
 35 # garbage
 36 rm -f *.o
```

The execution example of “build_readL2L_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2L_hdf5_f.sh
icc -g -c amsr2time_.c
ifort -g readL2L_hdf5.f amsr2time_.o -o readL2L_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.3.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2L_hdf5_f” is shown as follows.

```
$ ./readL2L_hdf5_f GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLA0000000.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_201303011809_125D_L2SGCLWLA000000
0
NumberOfScans(RETRIEVE BY ARRAY SIZE):       1972
OverlapScans(FIXED VALUE):          0
limit of NumberOfScans =  2200
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leaps
ec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time(scan=1): 2013/03/01 18:09:10
latlon(pixel=1,scan=1): (  84.4574, -78.1076)
geo1(pixel=1,scan=1): -32767.000 [Kg/m2] (PDQ:112)
```

8.4 Read L2 High Resolution Product

Precipitation (PRC) product is L2 High Resolution Product.

For Cloud Liquid Water(CLW), Sea Ice Concentration(SIC), Soil Moisture Content(SMC), Snow Depth(SND), Sea Surface Temperature(SST), and Sea Surface Wind Speed(SSW), and Total Precipitable Water (TPW), please refer to “7.3 Read L2 Low Resolution Product” on page 216.

8.4.1 Fortran sample (readL2H_hdf5.f) program

Sample program (readL2H_hdf5.f) which reads the metadata and the datasets of L2 is shown below. Values of data are dumped to the standard output.
Subroutines are prepared for the conversion of TAI93.

Metadata	Datasets
* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. * NumberOfScans are acquired from the array dimension of the dataset. Fixed values are used for the variable below. * OverlapScans	* Scan Time * Latitude of Observation Point for 89A - Latitude of Observation Point for 89B * Longitude of Observation Point for 89A - Longitude of Observation Point for 89B * Geophysical Data for 89A - Geophysical Data for 89B * Pixel Data Quality for 89A - Pixel Data Quality for 89B For details to P.20

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 4 C include
 5      include 'amsr2time_f.h'
 :
 8 C fixed value
 9      integer(4),parameter::LMT=2200 ! limit of NumberOfScans
10      integer(4),parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
11      integer(4),parameter::AM2_DEF_SNUM_LO=243 ! low  resolution pixel width
 :
14 C interface variable
15      integer(4) i,j          ! loop variable
16      integer(4) ret          ! return status
17      character(len=512) buf  ! text buffer
18      character(len=512) fn   ! filename
19      integer(HID_T) fhnd     ! file handle
20      integer(HID_T) ahnd     ! attribute handle
21      integer(HID_T) atyp     ! attribute type
22      integer(HID_T) dhnd     ! dataset handle
23      integer(HID_T) shnd     ! dataspace handle
24      integer(HSIZE_T) sz1(3) ! array size 1
25      integer(HSIZE_T) sz2(3) ! array size 2
 :
35      integer(4) num          ! NumberOfScans
36      integer(4) ovr          ! OverlapScans
37      parameter(ovr=0)
 :
41 C array data
42      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
43      real(4) lat89a(AM2_DEF_SNUM_HI,LMT)  ! lat for 89a
44      real(4) lat89b(AM2_DEF_SNUM_HI,LMT)  ! lat for 89b
45      real(4) lon89a(AM2_DEF_SNUM_HI,LMT)  ! lon for 89a
46      real(4) lon89b(AM2_DEF_SNUM_HI,LMT)  ! lon for 89b
47      real(4) geo1_89a(AM2_DEF_SNUM_HI,LMT)
48      real(4) geo1_89b(AM2_DEF_SNUM_HI,LMT)
49      integer(4) pdq1_89a(AM2_DEF_SNUM_HI,LMT)
50      integer(4) pdq1_89b(AM2_DEF_SNUM_HI,LMT)
```

Employ the USE statement to make use of the HDF5 functions in the program.

Include header files of subroutines for time conversion.

Limit of scan is sufficient in number 2200.
When you use Near real time operation product, you should set LMT=9000, because about length of 2 orbit may be stored in the products.

Define interface variables for HDF5.

Define the variables for metadata.
※Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. Fixed values are set in the program.

Use “AM2_COMMON_SCANTIME” data structure for the acquisition of scanning time.
Data dimensions vary with the respect to each products and datasets.
Limit of scan number is already defied in this program, because it also varies with products. (LMT=2200)
“AM2_DEF_SNUM_HI” is a value (486) defined in program, which is the number of observation points for each scan of high resolution data.
“AM2_DEF_SNUM_LO” is a value (243) defined in program, which is the number of observation points for each scan of low resolution data.

Define the variables for datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
59 C hdf5 initialize
60       call H5open_f(ret)
61       if(ret.lt.0)then
62         write(*,'(a,i12)')'h5open_f error: ',ret
63         call exit(1)
64       endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allows read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
65 C open
66       call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
67       if(ret.lt.0)then
68         write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
69         call exit(1)
70       endif
```

Specify NumberOfScans * Numbers written on the left are row number of the sample program.

Since the metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library, you need to acquire “NumberOfScans” from array dimension of the dataset “Scan Time”. First, open the dataset by using “H5Dopen_f” function. Next, get the dataspace type by using “H5Dget_space_f” function. Then you'll be able to acquire the array dimension of the dataset by using “H5Sget_simple_extent_dims_f” function.Close the dataset and dataspace identifier after you read the data.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
82 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
83 C calculate NumberOfScans from array size and OverlapScans
84       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
85       call H5Dget_space_f(dhnd,shnd,ret)
86       call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
87       if(ret.lt.0)then
88         write(*,'(a)')'H5Sget_simple_extent_dims error: Scan Time'
89         call exit(1)
90       endif
91       call H5Sclose_f(shnd,ret)
92       call H5Dclose_f(dhnd,ret)
93       num=sz1(1)-ovr*2
94       write(*,'(a,i12)')'NumberOfScans(RETRIEVE BY ARRAY SIZE): ',num
95       write(*,'(a,i12)')'OverlapScans(FIXED VALUE): ',ovr
```

NumberOfScans is represented as below.
Sum of scans in product - (OverlapScans x 2)

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Read scanning time * Numbers written on the left are row number of the sample program.

Open the dataset using H5Dopen_f function.
Dimension size of the dataset are required when reading.
TAI93 stored in scanning time data is converted to year, month, day, hour, minute, second, and millisecond.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
103 C read array: scantime
104       ! read
          call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
          sz1(1)=LMT
          call H5Dread_f(dhnd
         +,H5T_NATIVE_DOUBLE,r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
          if(ret.lt.0)then
            write(*,'(a,a)')'H5Dread error: ','Scan Time'
            call exit(1)
          endif
          call H5Dclose_f(dhnd,ret)
          ! convert
          call amsr2time(num,r8d1,st)
          ! sample display
          write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2,":",i2.2,":",i2.2)')
         +'time(scan=1): '
         +,st(1)%year
         +,st(1)%month
         +,st(1)%day
         +,st(1)%hour
         +,st(1)%minute
         +,st(1)%second
```

Read scanning time.

Time conversion.

Scanning time data is stored as TAI93 in AMSR2 product. TAI93 is the elapsed second time which includes the leap second from January 1st, 1993.
Subroutine for converting TAI93 to year, month, day, hour, minute, second, and millisecond is prepared in sample programs. This conversion is automatically applied when using AMTK. For details to P.14.

Time conversion.
amsr2time_(num,in,out)
num: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

Read latitude and longitude * Numbers written on the left are row number of the sample program.

Read latitude and longitude. → For details to P.20

```
125 C read array: latlon for 89a
126       ! read lat
127       call H5Dopen_f(fhnd
         +,'Latitude of Observation Point for 89A',dhnd,ret)
129       sz1(1)=LMT
130       sz1(2)=AM2_DEF_SNUM_HI
131       call H5Dread_f(dhnd
132      +,H5T_NATIVE_REAL,lat89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
133       if(ret.lt.0)then
134         write(*,'(a,a)')'H5Dread error: '
135      +  ,'Latitude of Observation Point for 89A'
136         call exit(1)
137       endif
138       call H5Dclose_f(dhnd,ret)
139       ! read lon
140       call H5Dopen_f(fhnd
141      +,'Longitude of Observation Point for 89A',dhnd,ret)
142       sz1(1)=LMT
143       sz1(2)=AM2_DEF_SNUM_HI
144       call H5Dread_f(dhnd
145      +,H5T_NATIVE_REAL,lon89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
146       if(ret.lt.0)then
147         write(*,'(a,a)')'H5Dread error: '
148      +  ,'Longitude of Observation Point for 89A'
149         call exit(1)
150       endif
151       call H5Dclose_f(dhnd,ret)
152       ! sample display
153       write(*,'(a,"(",f9.4,",",f9.4,")")')'latlon89a(pixel=1,scan=1): '
154      +,lat89a(1,1),lon89a(1,1)
```

Read latitude.

Read longitude.

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantity.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

Open an attribute.
call H5Aopen_f(dhnd,nam,ahnd,ret)
dhnd: Object id.
nam: Name of attribute.
ahnd: [Return value] Attribute id
ret: [Return value] Error: A negative value is returned.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

```
185 C read array: geophysical data for 1 layer for 89a
186       call H5Dopen_f(fhnd,'Geophysical Data for 89A',dhnd,ret)
187       ! get scale
188       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
189       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
190       call H5Aclose_f(ahnd,ret)
191       ! read
          sz1(1)=LMT
          sz1(2)=AM2_DEF_SNUM_HI
          call H5Dread_f(dhnd
         +,H5T_NATIVE_REAL,geo1_89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
          if(ret.lt.0)then
            write(*,'(a,a)')'H5Dread error: '
         +  ,'Geophysical Data for 89A'
            call exit(1)
          endif
          call H5Dclose_f(dhnd,ret)
          ! change scale
          do j=1,num
            do i=1,AM2_DEF_SNUM_HI
              if(geo1_89a(i,j).gt.-32767)geo1_89a(i,j)=geo1_89a(i,j)*sca
            enddo
          enddo
```

Read scale factor.

Read geophysical quantity.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value(-32767) when scale handling is applied.

Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

Read L2 pixel data quality.

```
231 C read array: pixel data quality for 1 layer for 89a
232       call H5Dopen_f(fhnd,'Pixel Data Quality for 89A',dhnd,ret)
233       sz1(1)=LMT
234       sz1(2)=AM2_DEF_SNUM_HI
235       call H5Dread_f(dhnd
236      +,H5T_NATIVE_INTEGER,pdq1_89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
237       if(ret.lt.0)then
238         write(*,'(a,a)')'H5Dread error: '
239      +  ,'Pixel Data Quality for 89A'
240         call exit(1)
241       endif
242       call H5Dclose_f(dhnd,ret)
```

L2 pixel data quality stores auxiliary information related to the calculation of geophysical quantities settled by the algorithm developers.
Value 0 to 15 shows good status, and 16 to 255 means bad status.
When the pixel value shows the bad status, Missing value (-32768) or Error value (-32761 to -32767) is stored in the geophysical quantity data.
Further information can be found on "AMSR2 Higher Level Product Format Specification"(*1).
(*1) http://suzaku.eorc.jaxa.jp/GCOM_W/data/data_w_format.html

Close HDF file * Numbers written on the left are row number of the sample program.

Close the HDF5 file.

```
261 C close
262       call H5Fclose_f(fhnd,ret)
263       call H5close_f(ret)
```

Terminate access to an HDF5 file.
call H5Fclose_f(fhnd,ret)
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
call H5close_f(ret)
ret: [Return value] Error: A negative value is returned.

8.4.2 Compile (Explanation of build_readL2H_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL2H_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11-12.
Since subroutine for time conversion is written in C language, specify C compiler.
Intel compiler (ifort/icc) or PGI compiler (pgf90/pgcc) is required.

```
10 # compiler
11 fc=ifort
12 cc=icc
13
14 # source filename
15 fsrc="readL2H_hdf5.f"
16 csrc="amsr2time_.c"
17 obj=`echo $csrc|sed "s/¥.c/.o/g"`
18
19 # output filename
20 out=readL2H_hdf5_f
21
22 # library order
23 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
24
25 # c compile
26 cmd="$cc -g -c $csrc"
27 echo $cmd
28 $cmd
29
30 # f compile
31 cmd="$fc -g $fsrc $obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
32 echo $cmd
33 $cmd
34
35 # garbage
36 rm -f *.o
```

The execution example of “build_readL2H_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL2H_hdf5_f.sh
icc -g -c amsr2time_.c
ifort -g readL2H_hdf5.f amsr2time_.o -o readL2H_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.4.3 Executions

Segmentation fault may occur because sample program contains many fixed arrays. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL2H_hdf5_f” is shown as follows.

```
$ ./readL2H_hdf5_f GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_201303011809_125D_L2SGPRCHA000000
0
NumberOfScans(RETRIEVE BY ARRAY SIZE):        1972
OverlapScans(FIXED VALUE):           0
limit of NumberOfScans =   2200
amsr2time: AMSR2_LEAP_DATA = /export/emc3/util/common/AMTK_AMSR2_DATA/leaps
ec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.00
amsr2time: year=1994 month= 7 tai93sec=   47174402.00
amsr2time: year=1996 month= 1 tai93sec=   94608003.00
amsr2time: year=1997 month= 7 tai93sec=  141868804.00
amsr2time: year=1999 month= 1 tai93sec=  189302405.00
amsr2time: year=2006 month= 1 tai93sec=  410227206.00
amsr2time: year=2009 month= 1 tai93sec=  504921607.00
amsr2time: year=2012 month= 7 tai93sec=  615254408.00
amsr2time: number of leap second = 8
time(scan=1): 2013/03/01 18:09:10
latlon89a(pixel=1,scan=1): (  84.4188, -77.9502)
latlon89b(pixel=1,scan=1): (  84.3305, -78.8925)
geo1_89a(pixel=1,scan=1): -32767.0 [mm/h] (PDQ: 16)
geo1_89b(pixel=1,scan=1): -32767.0 [mm/h] (PDQ: 16)
```

8.5 Read L3 Product [Brightness temperature]

8.5.1 Fortran sample program (readL3B_hdf5.f)

Sample program (readL3B_hdf5.f) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.	* Brightness Temperature (H) - Brightness Temperature (V)

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of AMTK will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 :
 9 C interface variable
10      integer(4) i,j         ! loop variable
11      integer(4) ret         ! return status
12      character(len=512) buf  ! text buffer
13      character(len=512) fn   ! filename
14      integer(HID_T) fhnd    ! file handle
15      integer(HID_T) ahnd    ! attribute handle
16      integer(HID_T) atyp    ! attribute type
17      integer(HID_T) dhnd    ! dataset handle
18      integer(HID_T) shnd    ! dataspace handle
19      integer(HSIZE_T) sz1(3) ! array size 1
20      integer(HSIZE_T) sz2(3) ! array size 2
21      integer(4) x           ! grid size x
22      integer(4) y           ! grid size y
 :
28 C meta data
29 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
30 C retrieve GranuleID from filename
31 C    character(len=512) geo  ! GeophysicalName
32      character(len=512) gid  ! GranuleID
 :
35 C array data
36      real(4),allocatable::tbH(:,:)
37      real(4),allocatable::tbV(:,:)
```

Employ the USE statement to make use of the HDF5 functions in the program.

Define interface variables for HDF5.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
46 C hdf5 initialize
47      call H5open_f(ret)
48      if(ret.lt.0)then
49        write(*,'(a,i12)')'h5open_f error: ',ret
50        call exit(1)
51      endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allows read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
52 C open
53      call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
54      if(ret.lt.0)then
55        write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
56        call exit(1)
57      endif
```

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension.
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
75 C get grid size
76       call H5Dopen_f(fhnd,'Brightness Temperature (H)',dhnd,ret)
77       call H5Dget_space_f(dhnd,shnd,ret)
78       call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
79       if(ret.lt.0)then
80         write(*,'(a,a)')'H5Sget_simple_extent_dims error: '
81      +  ,'Brightness Temperature (H)'
82         call exit(1)
83       endif
84       call H5Sclose_f(shnd,ret)
85       call H5Dclose_f(dhnd,ret)
86       x=sz1(1);
87       y=sz1(2);
88       write(*,'(a,i12)')'grid size x: ',x
89       write(*,'(a,i12)')'grid size y: ',y
```

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Allocate the memory.

```
90 C memory allocate
91       allocate(tbH(x,y),stat=ret)
92       if(ret.ne.0)then
93         write(*,'(a)')'memory allocate error: tbH'
94         call exit(1)
95       endif
```

Read brightness temperature * Numbers written on the left are row number of the sample program.

Brightness temperature is stored as 2 byte unsigned integer (0 to 65535). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (65535) and error value (65534) must be excluded.

Open an attribute.
call H5Aopen_f(dhnd,nam,ahnd,ret)
dhnd: Object id.
nam: Name of attribute.
ahnd: [Return value] Attribute id
ret: [Return value] Error: A negative value is returned.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
101 C read horizontal
102       call H5Dopen_f(fhnd,'Brightness Temperature (H)',dhnd,ret)
103       ! get scale
104       call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
105       call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
106       call H5Aclose_f(ahnd,ret)
1 7       ! read
          sz1(1)=y
          sz1(2)=x
1 0       call H5Dread_f(dhnd
1 1      +,H5T_NATIVE_REAL,tbH,sz1,ret,H5S_ALL_F,H5S_ALL_F)
1 2       if(ret.lt.0)then
1 3         write(*,'(a,a)')'H5Dread error: '
1 4      +  ,'Brightness Temperature (H)'
1 5         call exit(1)
          endif
1 7       call H5Dclose_f(dhnd,ret)
118       ! change scale
1 0       do j=1,y
            do i=1,x
1 1           if(tbH(i,j).lt.65534)tbH(i,j)=tbH(i,j)*sca
1 2         enddo
1 3       enddo
```

Read scale factor.

Read brightness temperature.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (65535) and error value (65534) when scale handling is applied.

Close HDF file * Numbers written on the left are row number of the sample program.

Release the memory.

```
239 C memory free
240       deallocate(tbH)
241       deallocate(tbV)
```

Close the HDF5 file.

Terminate access to an HDF5 file.
call H5Fclose_f(fhnd,ret)
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
call H5close_f(ret)
ret: [Return value] Error: A negative value is returned.

```
242 C close
243       call H5Fclose_f(fhnd,ret)
244       call H5close_f(ret)
245       end
```

8.5.2 Compile (Explanation of build_readL3B_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL3B_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (ifort) or PGI compiler (pgf90) are required.

```
10 # compiler
11 fc=ifort
12
13 # source filename
14 fsrc="readL3B_hdf5.f"
15
16 # output filename
17 out=readL3B_hdf5_f
18
19 # library order
20 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
21
22 # f compile
23 cmd="$fc -g $fsrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL3B_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3B_hdf5_f.sh
ifort -g readL3B_hdf5.f -o readL3B_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.5.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

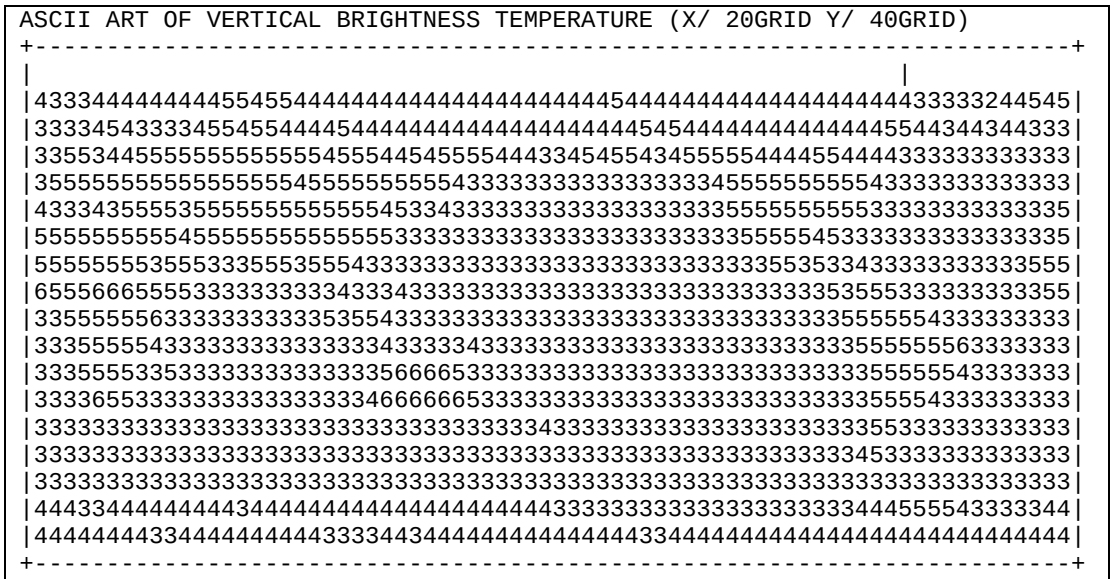
$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3B_hdf5_f” is shown as follows.

```
$ ./readL3B_hdf5_f GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_20130200_01M_EQMA_L3SGT06LA1110110
grid size x:       1440
grid size y:        720

ASCII ART OF HORIZONTAL BRIGHTNESS TEMPERATURE (X/ 20GRID Y/ 40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|411233344443444444444444444444444444444444444444444444433443222222444 4|
|111134311113444444444444444444444444444444444444444444444444434233343111|
|124523445555555444444444444444434312343444144555444444444442112111111111|
|154544444444444444444444444544111111111111111113554444444442221111111112|
|412242554414444544444445341131111111111111111111455544445511111111111115|
|444444444244455544555544111111111111111111111115555535111111111111111124|
|444444442444115551555311111111111111111121111111155151141111111111111444|
|555555555551111211113111211111111111111111111111111111141554111111111155|
|115555555111111111115155311111111111111111111111111111111455555411111111|
|111555553111111111111111421111312111111111111111111111111555555551111111|
|111555521511111111111111455555111111111111111111111111111145555521111111|
|111155511111111111111135555555111111111111111111111111111154552111111111|
|111111111111111111111111111111111131111111111111111111111155111111111111|
|111111111111111111111111111111111111111111111111111111111351111111111111|
|111111111111111111111111111111111111111111111111111111111111111111111111|
|343323444333342434334344444444334332111111111111111111344444441111133|
|333333333333333333333333333333343333333222223333334433443333333333333333|
+------------------------------------------------------------------------+
```

ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/ 20GRID Y/ 40GRID)



[#]:missing
[ ]:out of observation
[1]: 50-100K
[2]:100-150K
[3]:150-200K
[4]:200-250K
[5]:250-300K
[6]:300-350K
[*]:other

8.6 Read L3 Product [Geophysical quantity]
8.6.1 Fortran sample program (readL3G_hdf5.f)

Sample program (readL3G_hdf5.f) which reads the metadata and the datasets of L3 is shown below. Values of data are dumped to the standard output.

Metadata	Datasets
* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.	* Geophysical Data

In the following sample, we only explain how to read the data marked by * shown above. Explanation of similar description in the program will be skipped. Functions of HDF5 will be explained for the first time used in the program.

When only using HDF5 library, you have to open and close the metadata/datasets everytime after you open the HDF file. Flow for acquisition of data is shown below.

Open the metadata/datasets -> Acquire the scale factor (if necessary) -> Read the data -> Close metadata/datasets

Definition of variable * Numbers written on the left are row number of the sample program.

```
 1      program main
 2      use hdf5
 3      implicit none
 :
 9 C interface variable
10      integer(4) i,j          ! loop variable
11      integer(4) ret          ! return status
12      character(len=512) buf  ! text buffer
13      character(len=512) fn   ! filename
14      integer(HID_T) fhnd     ! file handle
15      integer(HID_T) ahnd     ! attribute handle
16      integer(HID_T) atyp     ! attribute type
17      integer(HID_T) dhnd     ! dataset handle
18      integer(HID_T) shnd     ! dataspace handle
19      integer(HSIZE_T) sz1(3) ! array size 1
20      integer(HSIZE_T) sz2(3) ! array size 2
21      integer(4) x            ! grid size x
22      integer(4) y            ! grid size y
 :
30 C meta data
31 C HDF5 FORTRAN LIBRARY CAN'T RETRIEVE VARIABLE STRING !
32 C retrieve GranuleID from filename
33 C    character(len=512) geo  ! GeophysicalName
34      character(len=512) gid  ! GranuleID
 :
37 C array data
38      real(4),allocatable::geo1(:,:)
39      real(4),allocatable::geo2(:,:)
40      real(4),allocatable::geotmp(:,:,:)
```

Employ the USE statement to make use of the HDF5 functions in the program.

Define interface variables for HDF5.

Define the variables for datasets.

At this time memory area of the variables for the datasets are not allocated.
Allocating memory will be held after researching the size of datasets.

Open HDF file * Numbers written on the left are row number of the sample program.

Initialize the HDF5 library.

Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.

```
49 C hdf5 initialize
50      call H5open_f(ret)
51      if(ret.lt.0)then
52        write(*,'(a,i12)')'h5open_f error: ',ret
53        call exit(1)
54      endif
```

Open the HDF5 file.

Open an existing HDF5 file.
call H5Fopen_f(fn,label1,fhnd,ret,label2)
fn: AMSR2 HDF file name.
label1: File access flags. H5F_ACC_RDONLY_F allow read-only access to file.
fhnd: [Return value] HDF access file id is returned.
ret: [Return value] Error: A negative value is returned.
label2: Use H5P_DEFAULT_F for default file access properties.

```
55 C open
56      call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
57      if(ret.lt.0)then
58        write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
59        call exit(1)
60      endif
```

Acquisition of the array size and memory allocation * Numbers written on the left are row number of the sample program.

Get the size of the array.

Open an existing dataset.
call H5Dopen_f(fhnd,nam,dhnd,ret)
fhnd: HDF access file id.
nam: The name of the dataset to access.
dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.

Return an identifier for the dataspace.
call H5Dget_space_f(dhnd,shnd,ret)
dhnd: Dataset id.
shnd: [Return value] Dataspace id.
ret: [Return value] Error: A negative value is returned.

Acquisition of data array dimension
call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
shnd: Dataspace id.
sz1: [Return value] Size of each dimension.
sz2: [Return value] Maximum size of each dimension.
ret: [Return value] Error: A negative value is returned.

```
79 C get grid size
 80      call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
 81      call H5Dget_space_f(dhnd,shnd,ret)
 82      call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)
 83      if(ret.lt.0)then
 84        write(*,'(a,a)')'H5Sget_simple_extent_dims error: '
 85     +  ,'Geophysical Data'
 86        call exit(1)
 87      endif
 88      call H5Sclose_f(shnd,ret)
 89      call H5Dclose_f(dhnd,ret)
 90      x=sz1(2);
 91      y=sz1(3);
 92      write(*,'(a,i12)')'grid size x: ',x
 93      write(*,'(a,i12)')'grid size y: ',y
```

Release the dataspace.
call H5Sclose_f(shnd,ret)
shnd: Dataspace id.
ret: [Return value] Error: A negative value is returned.

Close the dataset.
call H5Dclose_f(dhnd,ret)
dhnd: Dataset id.
ret: [Return value] Error: A negative value is returned.

Allocate the memory.

```
 94 C memory allocate layer 1
 95      allocate(geo1(x,y),stat=ret)
 96      if(ret.ne.0)then
 97        write(*,'(a)')'memory allocate error: geo1'
 98        call exit(1)
 99      endif
```

Read geophysical quantities * Numbers written on the left are row number of the sample program.

Read geophysical quantities.
Snow depth product and Sea surface temperature have 2 layer of geophysical quantities.
Geophysical quantity stored in the second layer of snow depth product is “Snow Water Equivalent”.
Geophysical quantity stored in the second layer of sea surface temperature product is “SST obtained by 10GHz”.

Geophysical quantity is stored as 2 byte signed integer (-32768 to 32767). To acquire its original value, reading scale factor which is stored with the brightness temperature data as the attribute is necessary.
Scale handling to missing value (-32768) and error value (-32767) must be excluded.

Open an attribute.
call H5Aopen_f(dhnd,nam,ahnd,ret)
dhnd: Object id.
nam: Name of attribute.
ahnd: [Return value] Attribute id
ret: [Return value] Error: A negative value is returned.

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.

Read an attribute.
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
otyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: Specify the array dimension of buf. (Ignored when buf is a scalar.)
ret: [Return value] Error: A negative value is returned.

Read raw data from a dataset.
call H5Dread_f(dhnd,dtyp,buf,sz1,ret,label1,label2)
dhnd: Dataset id.
dtyp: Datatype id. (Dataset is automatically converted to specified data type.)
buf: Variable storing acquired data.
sz1: Specify the array dimension of datasets.
ret: [Return value] Error: A negative value is returned.
label1,label2: Use when reading part of the array. Use H5S_ALL_F for both labels when reading entire array.

```
113 C read layer 1
114       if(gid(30:32).ne.'SND')then
115         call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
116         ! get scale
117         call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
118         call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
119         call H5Aclose_f(ahnd,ret)
1 0         ! read
            sz1(1)=y
1 2         sz1(2)=x
1 3         call H5Dread_f(dhnd
1 4       +  ,H5T_NATIVE_REAL,geo1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
1 5         if(ret.lt.0)then
1 6           write(*,'(a,a)')'H5Dread error: '
1 7       +    ,'Geophysical Data'
1 8           call exit(1)
1 9         endif
1 0         call H5Dclose_f(dhnd,ret)
1 1         ! change scale
1 2         do j=1,y
1 3           do i=1,x
1 4             if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
1 5           enddo
            enddo
1 7       endif
```

Read scale factor.

Read geophysical quantity.

Scale handling is automatically applied when using AMTK. For details to P.22.

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```
138 C read layer 1 & 2
139       if(gid(30:32).eq.'SND')then
140         call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
141         ! get scale
142         call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
143         call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
144         call H5Aclose_f(ahnd,ret)
145         ! read
146         sz1(1)=y
147         sz1(2)=x
148         sz1(3)=2
149         call H5Dread_f(dhnd
150      +  ,H5T_NATIVE_REAL,geotmp,sz1,ret,H5S_ALL_F,H5S_ALL_F)
151         if(ret.lt.0)then
152           write(*,'(a,a)')'H5Dread error: '
153      +    ,'Geophysical Data'
154           call exit(1)
155         endif
156         call H5Dclose_f(dhnd,ret)
157         ! separate
158         do j=1,y
159           do i=1,x
160             geo1(i,j)=geotmp(1,i,j)
161             geo2(i,j)=geotmp(2,i,j)
162           enddo
163         enddo
164         ! change scale
165         do j=1,y
166           do i=1,x
167             if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
168             if(geo2(i,j).gt.-32767)geo2(i,j)=geo2(i,j)*sca
169           enddo
170         enddo
171       endif
```

Close HDF file

* Numbers written on the left are row number of the sample program.

Release the memory.

```
357 C memory free
358       deallocate(geo1)
359       if(gid(30:32).eq.'SND')then
360         deallocate(geo2)
361       endif
```

Close the HDF5 file.

Terminate access to an HDF5 file.
call H5Fclose_f(fhnd,ret)
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.

Flush all data to disk, close all open identifiers, and clean up memory.
call H5close_f(ret)
ret: [Return value] Error: A negative value is returned.

```
362 C close
363       call H5Fclose_f(fhnd,ret)
364       call H5close_f(ret)
```

8.6.2 Compile (Explanation of build_readL3G_hdf5_f.sh)

We explain how to compile the Fortran program by using script “build_readL3G_hdf5_f.sh”.

* Numbers written on the left are row number of the sample program.

```
 1 #!/bin/sh
 2
 3 ### environment
 4 export LANG=C
 5
```

Specify the library directories in row number 7-8.
“include” and “lib” directories are necessary under the each library directories.

```
 6 # library directory
 7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
 8 SZIP=/home/user1/util/szip_2.1
 9
```

Specify the compiler you use in row number 11.
Intel compiler (ifort) or PGI compiler (pgf90) are required.

```
10 # compiler
11 fc=ifort
12
13 # source filename
14 fsrc="readL3G_hdf5.f"
15
16 # output filename
17 out=readL3G_hdf5_f
18
19 # library order
20 lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"
21
22 # f compile
23 cmd="$fc -g $fsrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib
-L$SZIP/lib $lib"
24 echo $cmd
25 $cmd
26
27 # garbage
28 rm -f *.o
```

The execution example of “build_readL3G_hdf5_f.sh” is shown in the following.
* Line feeds are inserted for convenience.

```
$ ./build_readL3G_hdf5_f.sh
ifort -g readL3G_hdf5.f -o readL3G_hdf5_f
-I/home/user1/util/hdf5_1.8.4-patch1/include
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm
```

8.6.3 Executions

Segmentation fault due to the lack of resources may occur. When it happens, please type the following command to avoid it.

< For csh or tcsh >	< For sh or bash >

$ unlimit	* Type the following four commands in order.
	$ ulimit -d unlimited
	$ ulimit -m unlimited
	$ ulimit -s unlimited
	$ ulimit -v unlimited

The example of executing “readL3G_hdf5_f” is shown as follows.

```
$ ./readL3G_hdf5_f GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_20130200_01M_EQMA_L3SGCLWLA0000000
grid size x:       1440
grid size y:        720

ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/ 20GRID Y/ 40GRID)
+------------------------------------------------------------------------+
|                                                                        |
|#21#####################################################################|
|2212###1113#############################################################222|
|1#################################11######3###############011#42223322|
|2#############################0010111241333323#############343413131#|
|#20#######2###############00#12234523442132111 20#########3355421131121#|
|#########################4#32222232321001101111 10#######14*41111201111##|
|########0###000###0####1111111210120111 0*131121100##1#11#01211110100###|
|###########1011111111#211#11111210011121333223224321000#1###1111000000##|
|11#######00010122125#3###1#12232321100010020110001011122#######222321111|
|000######122131112311112#01112#113122123231101000000000110########1001110|
|100####32#12111141111000######223232242121244133111001101 0######11111100|
|1000###12112322112511 00########131112232110121313122011010#####231111001|
|11131212231110111111211111110012102#123122122331121311102 1##011124223242|
|*33222511232222222232233314210224223222110242222221221 13##0111421221132|
|10112211222111122211111111122231223223233124332121232322122331223222211 1|
|#######################################3111221222211121122 1########21221##|
|#####################################################################|
+------------------------------------------------------------------------+

[#]:missing
[ ]:out of observation
[0]:  0.000 -  0.030 Kg/m2
[1]:  0.030 -  0.060 Kg/m2
[2]:  0.060 -  0.090 Kg/m2
[3]:  0.090 -  0.120 Kg/m2
[4]:  0.120 -  0.150 Kg/m2
[5]:  0.150 -  0.180 Kg/m2
[*]:other
```

9. AMSR2 Product List

Dataset used in sample programs are shown in Table 10 to 15. Apart from these, more datasets are stored in AMSR2 product. For details to “AMSR2 Product I/O Toolkit (AMTK), User's Manual” which can be able to download from “GCOM-W1 Data Providing Service” (http://gcom-w1.jaxa.jp/).

Table 10. L1B datasets used in the sample programs

Data Name Datasets marked by * are not stored in HDF5	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Scan Time	R8	TM	AMTK_getScanTime()		SCAN
Latitude of Observation Point for 89A	R4	LL	AMTK_getLatLon()	AM2_LATLON_89A	486 x SCAN
Longitude of Observation Point for 89A	R4				
Latitude of Observation Point for 89B	R4	LL	AMTK_getLatLon()	AM2_LATLON_89B	486 x SCAN
Longitude of Observation Point for 89B	R4				
* Latitude (Low mean)		LL	AMTK_getLatLon()	AM2_LATLON_LO	243 x SCAN
* Longitude (Low mean)					
* Latitude (6G)		LL	AMTK_getLatLon()	AM2_LATLON_06	243 x SCAN
* Longitude (6G)					
* Latitude (7G)		LL	AMTK_getLatLon()	AM2_LATLON_07	243 x SCAN
* Longitude (7G)					
* Latitude (10G)		LL	AMTK_getLatLon()	AM2_LATLON_10	243 x SCAN
* Longitude (10G)					
* Latitude (18G)		LL	AMTK_getLatLon()	AM2_LATLON_18	243 x SCAN
* Longitude (18G)					
* Latitude (23G)		LL	AMTK_getLatLon()	AM2_LATLON_23	243 x SCAN
* Longitude (23G)					
* Latitude (36G)		LL	AMTK_getLatLon()	AM2_LATLON_36	243 x SCAN
* Longitude (36G)					
Brightness Temperature (6.9GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB06H	243 x SCAN
Brightness Temperature (6.9GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB06V	243 x SCAN
Brightness Temperature (7.3GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB07H	243 x SCAN
Brightness Temperature (7.3GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB07V	243 x SCAN
Brightness Temperature (10.7GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB10H	243 x SCAN
Brightness Temperature (10.7GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB10V	243 x SCAN
Brightness Temperature (18.7GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB18H	243 x SCAN
Brightness Temperature (18.7GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB18V	243 x SCAN
Brightness Temperature (23.8GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB23H	243 x SCAN
Brightness Temperature (23.8GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB23V	243 x SCAN
Brightness Temperature (36.5GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB36H	243 x SCAN
Brightness Temperature (36.5GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB36V	243 x SCAN
Brightness Temperature (89.0GHz-A,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89AH	486 x SCAN
Brightness Temperature (89.0GHz-A,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89AV	486 x SCAN
Brightness Temperature (89.0GHz-B,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89BH	486 x SCAN
Brightness Temperature (89.0GHz-B,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89BV	486 x SCAN
Pixel Data Quality 6 to 36	U1	I4	AMTK_get_SwathInt()	AM2_PIX_QUAL_LO	486 x SCAN
Pixel Data Quality 89	U1	I4	AMTK_get_SwathInt()	AM2_PIX_QUAL_HI	486 x SCAN
Land_Ocean Flag 6 to 36	U1	I4	AMTK_get_SwathInt()	AM2_LOF_LO	243 x SCAN x 6
Land_Ocean Flag 89	U1	I4	AMTK_get_SwathInt()	AM2_LOF_HI	486 x SCAN x 2
Earth Incidence	I2	R4	AMTK_get_SwathFloat()	AM2_EARTH_INC	243 x SCAN
Earth Azimuth	I2	R4	AMTK_get_SwathFloat()	AM2_EARTH_AZ	243 x SCAN

I4:4byte signed integer, I2:2byte signed integer, U2:2byte unsigned integer, U1:1byte unsigned integer, R8:8byte double precision, R4:4byte single precision, TM:“AM2_COMMON_SCANTIME” data structure, LL:“AM2_COMMON_LATLON” data structure.

Table 11. L1R datasets used in the sample programs

Data Name Datasets marked by * are not stored in HDF5	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Scan Time	R8	TM	AMTK_getScanTime()		SCAN
Latitude of Observation Point for 89A	R4	LL	AMTK_getLatLon()	AM2_LATLON_RS_89A	486 x SCAN
Longitude of Observation Point for 89A	R4				
Latitude of Observation Point for 89B	R4	LL	AMTK_getLatLon()	AM2_LATLON_RS_89B	486 x SCAN
Longitude of Observation Point for 89B	R4				
* Latitude (For low frequency data)		LL	AMTK_getLatLon()	AM2_LATLON_RS_LO	243 x SCAN
* Longitude (For low frequency data)					
Brightness Temperature (res06,6.9GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES06_TB06H	243 x SCAN
Brightness Temperature (res06,6.9GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES06_TB06V	243 x SCAN
Brightness Temperature (res06,7.3GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES06_TB07H	243 x SCAN
Brightness Temperature (res06,7.3GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES06_TB07V	243 x SCAN
Brightness Temperature (res10,10.7GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES10_TB10H	243 x SCAN
Brightness Temperature (res10,10.7GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES10_TB10V	243 x SCAN
Brightness Temperature (res23,18.7GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES23_TB18H	243 x SCAN
Brightness Temperature (res23,18.7GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES23_TB18V	243 x SCAN
Brightness Temperature (res23,23.8GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES23_TB23H	243 x SCAN
Brightness Temperature (res23,23.8GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES23_TB23V	243 x SCAN
Brightness Temperature (res36,36.5GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES36_TB36H	243 x SCAN
Brightness Temperature (res36,36.5GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES36_TB36V	243 x SCAN
Brightness Temperature (res36,89.0GHz,H)	U2	R4	AMTK_get_SwathFloat()	AM2_RES36_TB89H	243 x SCAN
Brightness Temperature (res36,89.0GHz,V)	U2	R4	AMTK_get_SwathFloat()	AM2_RES36_TB89V	243 x SCAN
Brightness Temperature (original,89GHz-A,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89AH	486 x SCAN
Brightness Temperature (original,89GHz-A,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89AV	486 x SCAN
Brightness Temperature (original,89GHz-B,H)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89BH	486 x SCAN
Brightness Temperature (original,89GHz-B,V)	U2	R4	AMTK_get_SwathFloat()	AM2_TB89BV	486 x SCAN
Pixel Data Quality 6 to 36	U1	I4	AMTK_get_SwathInt()	AM2_PIX_QUAL_LO	486 x SCAN
Pixel Data Quality 89	U1	I4	AMTK_get_SwathInt()	AM2_PIX_QUAL_HI	486 x SCAN
Land_Ocean Flag 6 to 36	U1	I4	AMTK_get_SwathInt()	AM2_LOF_RES_LO	243 x SCAN x 4
Land_Ocean Flag 89	U1	I4	AMTK_get_SwathInt()	AM2_LOF_RES_HI	486 x SCAN x 2
Earth Incidence	I2	R4	AMTK_get_SwathFloat()	AM2_EARTH_INC	243 x SCAN
Earth Azimuth	I2	R4	AMTK_get_SwathFloat()	AM2_EARTH_AZ	243 x SCAN

I4: 4byte signed integer, I2:2byte signed integer, U2: 2byte unsigned integer, U1: 1byte unsigned integer, R8: 8byte double precision, R4: 4byte single precision, TM: “AM2_COMMON_SCANTIME” data structure, LL: “AM2_COMMON_LATLON” data structure.

Table 12. L2 low resolution data used in the sample programs

Data Name	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Scan Time	R8	TM	AMTK_getScanTime()		SCAN
Latitude of Observation Point	R4	LL	AMTK_getLatLon()	AM2_LATLON_L2_LO	243 x SCAN
Longitude of Observation Point	R4				
Geophysical Data	I2	R4	AMTK_get_SwathFloat()	AM2_SWATH_GEOA (*1)	LAYER x 243 x SCAN
Pixel Data Quality	U1	U1	AMTK_get_SwathUChar()	AM2_PIX_QUAL	243 x SCAN x LAYER

I4: 4byte signed integer, I2: 2byte signed integer, U2: 2byte unsigned integer, U1: 1byte unsigned integer, R8: 8byte double precision, R4: 4byte single precision, TM: “AM2_COMMON_SCANTIME” data structure, LL: “AM2_COMMON_LATLON” data structure.
(*1) Access label “AM2_SWATH_GEO1” and “AM2_SWATH_GEO2” inputs and outputs 2-dimensions data of first and second layer of the datasets.

Table 13. L2 high resolution data used in the sample programs

Data Name	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Scan Time	R8	TM	AMTK_getScanTime()		SCAN
Latitude of Observation Point for 89A	R4	LL	AMTK_getLatLon()	AM2_LATLON_L2_89A	486 x SCAN
Longitude of Observation Point for 89A	R4				
Latitude of Observation Point for 89B	R4	LL	AMTK_getLatLon()	AM2_LATLON_L2_89B	486 x SCAN
Longitude of Observation Point for 89B	R4				
Geophysical Data for 89A	I2	R4	AMTK_get_SwathFloat()	AM2_SWATHA_GEOA (*1)	LAYER x 486 x SCAN
Geophysical Data for 89B	I2	R4	AMTK_get_SwathFloat()	AM2_SWATHB_GEOA (*2)	LAYER x 486 x SCAN
Pixel Data Quality for 89A	U1	U1	AMTK_get_SwathUChar()	AM2_PIX_QUAL_A	486 x SCAN x LAYER
Pixel Data Quality for 89B	U1	U1	AMTK_get_SwathUChar()	AM2_PIX_QUAL_B	486 x SCAN x LAYER

I4: 4byte signed integer, I2: 2byte signed integer, U2: 2byte unsigned integer, U1: 1byte unsigned integer, R8: 8byte double precision, R4: 4byte single precision, TM: “AM2_COMMON_SCANTIME” data structure, LL: “AM2_COMMON_LATLON” data structure.
(*1) Access label “AM2_SWATHA_GEO1” and “AM2_SWATHA_GEO2” inputs and outputs 2-dimensions data of first and second layer of the datasets.
(*2) Access label “AM2_SWATHB_GEO1” and “AM2_SWATHB_GEO2” inputs and outputs 2-dimensions data of first and second layer of the datasets.

Table 14. L3 data [Brightness Temperature] used in the sample programs

Data Name	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Brightness Temperature (H)	U2	R4	AMTK_get_GridFloat()	AM2_GRID_TBH	(*1)
Brightness Temperature (V)	U2	R4	AMTK_get_GridFloat()	AM2_GRID_TBV	(*1)

I4:4byte signed integer, I2:2byte signed integer, U2:2byte unsigned integer, U1:1byte unsigned integer, R8:8byte double precision, R4:4byte single precision, TM: “AM2_COMMON_SCANTIME” data structure, LL: “AM2_COMMON_LATLON” data structure.
(*1) For details to P.13, Figure 6. L3 granule ID.

Table 15. L3 data [Geophysical quantity] used in the sample programs

Data Name	HDF5 Output	AMTK Output	AMTK Access Function	AMTK Access Label	Size
Geophysical Data	I2	R4	AMTK_get_GridFloat()	AM2_GRID_GEOA (*2)	(*1) x LAYER

I4:4byte signed integer, I2:2byte signed integer, U2:2byte unsigned integer, U1:1byte unsigned integer, R8:8byte double precision, R4:4byte single precision, TM: “AM2_COMMON_SCANTIME” data structure, LL: “AM2_COMMON_LATLON” data structure.
(*1) For details to P.13, Figure 6. L3 granule ID.
(*2) Access label “AM2_GRID_GEO1” and “AM2_GRID_GEO2” inputs and outputs 2-dimensions data of first and second layer of the datasets.

10. Parameters Defined in AMTK

Parameters Defined in AMTK are shown in Table 16 and 17.

Table 16. AMTK defined parameters

Description	Name	Value
Number of observation points for each scan of high resolution data.	AM2_DEF_SNUM_HI	486
Number of observation points for each scan of low resolution data.	AM2_DEF_SNUM_LO	243
2byte signed integer loss value	AM2_DEF_IMISS	-32768
2byte unsigned integer loss value	AM2_DEF_UIMISS	65535
1byte unsigned integer loss value	AM2_DEF_CMISS	255
4byte real loss value	AM2_DEF_RMISS	-9999.0

Table 17. AMTK defined L3 parameters (number of pixels for each projection)

Projection	Direction [Number]	Name	Value
Equi-rectangular projection (Low resolution)	Longitude	AM2_DEF_L3L_EQ_X	1440
	Latitude	AM2_DEF_L3L_EQ_Y	720
Equi-rectangular projection (High resolution)	Longitude	AM2_DEF_L3H_EQ_X	3600
	Latitude	AM2_DEF_L3H_EQ_Y	1800
Northern polar stereo projection (Low resolution, TB/SIC)	Width	AM2_DEF_L3L_PN1_X	304
	Length	AM2_DEF_L3L_PN1_Y	448
Northern polar stereo projection (High resolution, TB/SIC)	Width	AM2_DEF_L3H_PN1_X	760
	Length	AM2_DEF_L3H_PN1_Y	1120
Southern polar stereo projection (Low resolution, TB/SIC)	Width	AM2_DEF_L3L_PS_X	316
	Length	AM2_DEF_L3L_PS_Y	332
Southern polar stereo projection (High resolution, TB/SIC)	Width	AM2_DEF_L3H_PS_X	790
	Length	AM2_DEF_L3H_PS_Y	830
Northern polar stereo projection (Low resolution, SND)	Width	AM2_DEF_L3L_PN2_X	432
	Length	AM2_DEF_L3L_PN2_Y	574
Northern polar stereo projection (High resolution, SND)	Width	AM2_DEF_L3H_PN2_X	1080
	Length	AM2_DEF_L3H_PN2_Y	1435