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


Japan Aerospace Exploration Agency
Earth Observation Research Center
## Change record

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

1. How to Get Related Manuals and Products ................................................................. 4
2. Overview of this Data Users Manual ............................................................................. 5
3. Basic Knowledge of GCOM-W1/AMSR2 Products .................................................... 6
   3.1 Satellite Orbit ........................................................................................................... 6
   3.2 Granules ............................................................................................................... 9
   3.3 Level 1 and Higher Level Products ..................................................................... 11
   3.4 File Name Convention ......................................................................................... 12
   3.5 Scans and Overlaps ............................................................................................ 13
   3.6 Scan Time ............................................................................................................ 14
   3.7 Frequency and Observation Footprint ................................................................. 14
   3.8 Level 1 Resampling Products (L1R) .................................................................... 18
   3.9 Land/Ocean Flag ................................................................................................ 18
   3.10 Latitude and Longitude Stored in L1 and L2 Products ....................................... 20
   3.11 Altitude Correction Processed in L1R Products .................................................. 21
   3.12 Data Set Specification ....................................................................................... 22
4. Installation of Libraries ............................................................................................... 24
5. C Programming with AMTK ....................................................................................... 31
   5.1 Read L1B Product ............................................................................................... 31
   5.2 Read L1R Product .............................................................................................. 41
   5.3 Read L2 Low Resolution Product ...................................................................... 51
   5.4 Read L2 High Resolution Product ................................................................... 60
   5.5 Read L3 Product [Brightness temperature] ......................................................... 68
   5.6 Read L3 Product [Geophysical quantity] ............................................................ 75
6. FORTRAN90 Programming with AMTK .................................................................. 81
   6.1 Read L1B Product ............................................................................................. 81
   6.2 Read L1R product ............................................................................................ 91
   6.3 Read L2 Low Resolution Product .................................................................... 101
   6.4 Read L2 High Resolution Product .................................................................. 110
   6.5 Read L3 Product [Brightness temperature] ....................................................... 118
   6.6 Read L3 Product [Geophysical quantity] ......................................................... 125
7. C Programming with HDF5 library (without AMTK) ................................................. 131
   7.1 Read L1B Product ........................................................................................... 131
   7.2 Read L1R Product .......................................................................................... 144
   7.3 Read L2 Low Resolution Product .................................................................. 156
   7.4 Read L2 High Resolution Product .................................................................. 166
   7.5 Read L3 Product [Brightness temperature] ....................................................... 175
   7.6 Read L3 Product [Geophysical quantity] .......................................................... 183
8. FORTRAN90 Programming with HDF5 library (without AMTK) .......................... 191
   8.1 Read L1B Product ............................................................................................ 191
   8.2 Read L1R Product ........................................................................................ 204
   8.3 Read L2 Low Resolution Product .................................................................. 216
   8.4 Read L2 High Resolution Product ................................................................ 228
   8.5 Read L3 Product [Brightness temperature] ....................................................... 238
   8.6 Read L3 Product [Geophysical quantity] .......................................................... 246
9. AMSR2 Product List .................................................................................................. 253
10. Parameters Defined in AMTK .................................................................................. 256
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

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

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>
<thead>
<tr>
<th>Geophysical Parameters Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Precipitable Water (TPW)</td>
</tr>
<tr>
<td>Cloud Liquid Water (CLW)</td>
</tr>
<tr>
<td>Precipitation (PRC)</td>
</tr>
<tr>
<td>Sea Surface Temperature (SST)</td>
</tr>
<tr>
<td>Sea Surface Wind Speed (SSW)</td>
</tr>
<tr>
<td>Sea Ice Concentration (SIC)</td>
</tr>
<tr>
<td>Snow Depth (SND)</td>
</tr>
<tr>
<td>Soil Moisture Content (SMC)</td>
</tr>
</tbody>
</table>

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.
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.

<table>
<thead>
<tr>
<th>Byte Location</th>
<th>123456789012345678901234567890123456789012345678901</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMSR2</td>
<td>GW1AM2_YYYYMMDDHHmmPPPxLLxKKKrdvaaaappp</td>
</tr>
<tr>
<td>L1 Sample</td>
<td>GW1AM2_201209071216_068D_L1SGBTBR_0000000</td>
</tr>
<tr>
<td>L2 Sample</td>
<td>GW1AM2_201209071216_068D_L2SGSSTLA0000000</td>
</tr>
</tbody>
</table>

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
Figure 5. 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 a 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.
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>
<thead>
<tr>
<th>No.</th>
<th>Frequency/polarization</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.9GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6.9GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7.3GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7.3GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10.7GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10.7GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>18.7GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>18.7GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>23.8GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>23.8GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>36.5GHz horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>36.5GHz vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>89.0GHz A horizontal polarized channel</td>
<td>Low Frequency Channels: Number of observation points are 243.</td>
</tr>
<tr>
<td>14</td>
<td>89.0GHz A vertical polarized channel</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>89.0GHz B horizontal polarized channel</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>89.0GHz B vertical polarized channel</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Footprint (along scan x along track)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9GHz</td>
<td>35km x 62km</td>
</tr>
<tr>
<td>7.3GHz</td>
<td>34km x 58km</td>
</tr>
<tr>
<td>10.7GHz</td>
<td>24km x 42km</td>
</tr>
<tr>
<td>18.7GHz</td>
<td>14km x 22km</td>
</tr>
<tr>
<td>23.8GHz</td>
<td>15km x 26km</td>
</tr>
<tr>
<td>36.5GHz</td>
<td>7km x 12km</td>
</tr>
<tr>
<td>89.0GHz</td>
<td>3km x 5km</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Footprint</th>
<th>6G (H/V) 243 points</th>
<th>7G (H/V) 243 points</th>
<th>10G (H/V) 243 points</th>
<th>18G (H/V) 243 points</th>
<th>23G (H/V) 243 points</th>
<th>36G (H/V) 243 points</th>
<th>89G (H/V) 248 points</th>
<th>89G A-horn (H/V) 486 points</th>
<th>89G B-horn (H/V) 486 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>6GHz</td>
<td>☆</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>10GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>△</td>
<td>△</td>
</tr>
<tr>
<td>36GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☆</td>
<td>○</td>
<td></td>
</tr>
<tr>
<td>89GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>△</td>
</tr>
</tbody>
</table>

○ 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.
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 data 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 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

<table>
<thead>
<tr>
<th>Product</th>
<th>Stored latitude and longitude</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1B</td>
<td>Latitude and longitude of Observation Point for 89A</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Latitude and longitude of Observation Point for 89B</td>
<td></td>
</tr>
<tr>
<td>L1R</td>
<td>Latitude and longitude of Observation Point for 89A</td>
<td>Only odd points of 89GHz-A (origin 1) are used for low frequency data.</td>
</tr>
<tr>
<td></td>
<td>Latitude and longitude of Observation Point for 89B</td>
<td></td>
</tr>
<tr>
<td>L2 (L1B Input)</td>
<td>Average value of L1B latitude and longitude of each low frequency channels.</td>
<td></td>
</tr>
<tr>
<td>L2 (L1R Input)</td>
<td>Latitude and longitude of L1R low frequency (Odd points of 89GHz-A)</td>
<td></td>
</tr>
<tr>
<td>L2 (Precipitation)</td>
<td>Latitude and longitude of Observation Point for 89A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Latitude and longitude of Observation Point for 89B</td>
<td></td>
</tr>
</tbody>
</table>

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.

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 \( \theta \) as follows.

\[
L = H \tan \theta
\]

Figure 12. Altitude Correction
3.12 Data Set Specification

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

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

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)

- **Missing value (White; -32768)**
  - Geophysical parameters in land area of CLW products are not calculated.

- **Error value (Grey; -32767)**
  - Area outside of observation swath is distinguished from Missing value.
4. Installation of Libraries

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

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Test environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Intel(R) Xeon(R) CPU E5504</td>
</tr>
<tr>
<td>OS</td>
<td>Red Hat Enterprise Linux release 5.4</td>
</tr>
<tr>
<td>C compiler</td>
<td>GNU C compiler 4.1.2</td>
</tr>
<tr>
<td></td>
<td>Intel C compiler Version 11.1</td>
</tr>
<tr>
<td></td>
<td>PGI C compiler Version 10.9</td>
</tr>
<tr>
<td>FORTRAN compiler</td>
<td>Intel FORTRAN compiler Version 11.1</td>
</tr>
<tr>
<td></td>
<td>PGI FORTRAN compiler Version 10.9</td>
</tr>
<tr>
<td>HDF5</td>
<td>HDF5 version 1.8.4-patch1</td>
</tr>
<tr>
<td>SZIP</td>
<td>SZIP version 2.1</td>
</tr>
<tr>
<td>AMTK</td>
<td>version 1.11</td>
</tr>
</tbody>
</table>

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 -xzf 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.
(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 uncompress 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

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

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (7.3GHz,V)</td>
</tr>
<tr>
<td>Data calculated from observation point of 89GHz-A</td>
<td>* Brightness Temperature (10.7GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (Low mean)</td>
<td>- Brightness Temperature (10.7GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (6G)</td>
<td>- Brightness Temperature (18.7GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (7G)</td>
<td>- Brightness Temperature (18.7GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (10G)</td>
<td>- Brightness Temperature (23.8GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (18G)</td>
<td>- Brightness Temperature (23.8GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (23G)</td>
<td>- Brightness Temperature (36.5GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (36G)</td>
<td>- Brightness Temperature (36.5GHz,V)</td>
</tr>
</tbody>
</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

For details to P.20
AMSR2 data users manual

- 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.
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.

```
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.

```
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);
```

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

```c
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.

For details to P.14

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

```c
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.

For details to P.20
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

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.
```

```
// read array: pixel data quality for low
vpnt=pdqlo;
ret=AMTK_get_SwathInt(hnd,(int **)vpnt,1,num,AM2_PIX_QUAL_LO);
if(ret<0){
  printf("AMTK_get_SwathInt error: AM2_PIX_QUAL_LO\n");
  printf("amtk status: %d\n",ret);
  exit(1);
}
```

“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.

```
for(j=0;j<num;++j){
  for(i=0;i<AM2_DEF_SNUM_LO;++i){
    pdq06v[j][i]=pdqlo[j][(i*16+1)*10+pdqlo[j][i*16+0]];
    pdq06h[j][i]=pdqlo[j][(i*16+3)*10+pdqlo[j][i*16+2]];
    pdq07v[j][i]=pdqlo[j][(i*16+5)*10+pdqlo[j][i*16+4]];
    pdq07h[j][i]=pdqlo[j][(i*16+7)*10+pdqlo[j][i*16+6]];
  }
}
```
Read L1B land ocean flag * Numbers written on the left are row number of the sample program.

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.

```
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.

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.

```
ret=AMTK_closeH5(hnd)
```

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.

```sh
#!/bin/sh

### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc=readL1B_amtk.c

# output filename
out=readL1B_amtk_c

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
cmd="$cc -g $csrc -o $out -I$AMTK/include -I$HDF5/include -I$SZIP/include
-LSAMTK/lib -LSHDF5/lib -LSZIP/lib $lib"

$ cmd

data cleanup
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.

```bash
$ ./readL1B_amtk_c_GW1AM2_201207261145_055A_L1S_GBTR_00000000.h5
input file: GW1AM2_201207261145_055A_LISGBTR_00000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_LISGBTR_0000000
ObservationStartDate: 2012-07-26T11:45:43.0182
EquatorCrossingDateTime: 2012-07-26T12:12:37.848
ObservationEndDate: 2012-07-26T12:35:09.73523
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G-1.25000,7G-1.0000,8G-1.25000,18G-1.25000,36G-1.0000,00
CoRegistrationParameterA2: 6G-0.00000,7G-0.10000,8G-0.30000,18G-0.10000,36G-0.00000,00
00
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)
lonlonlm[scan=0][pixel=0]: (-73.3538, 136.6228)
lonlon06[scan=0][pixel=0]: (-73.3592, 136.6213)
lonlon07[scan=0][pixel=0]: (-73.3497, 136.6429)
lonlon10[scan=0][pixel=0]: (-73.3506, 136.6001)
lonlon18[scan=0][pixel=0]: (-73.3592, 136.6213)
lonlon23[scan=0][pixel=0]: (-73.3506, 136.6001)
lonlon36[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]: 207.94
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]: 163.76
tb89ah[scan=0][pixel=0]: 179.27
tb89av[scan=0][pixel=0]: 179.27
tb89ah[scan=0][pixel=0]: 188.16
tb89av[scan=0][pixel=0]: 188.16
```

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

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (res06,6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (res06,6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (res06,7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (res06,7.3GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,V)</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality 6 to 36</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality 89</td>
</tr>
<tr>
<td></td>
<td>* Land_Ocean Flag 6 to 36</td>
</tr>
<tr>
<td></td>
<td>* Land_Ocean Flag 89</td>
</tr>
<tr>
<td></td>
<td>* Earth Incidence</td>
</tr>
<tr>
<td></td>
<td>- Earth Azimuth</td>
</tr>
</tbody>
</table>

Data extracted from observation point of 89GHz-A
- Latitude and longitude of low frequency 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, 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
AMS R2 data users manual

- 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.
AMS2 data users manual

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 prml[512]; // CoRegistrationParameterA1
26   char prml[512]; // CoRegistrationParameterA2
27
28   // array data
29   AM2_COMMON_SCANTIME st[LMT]; // scantime
30   AM2_COMMON_LATLON ll89ar[LMT][AM2_DEF_SNUM_HI]; // latlon for 89a altitude
31   AM2_COMMON_LATLON ll89br[LMT][AM2_DEF_SNUM_HI]; // latlon for 89b altitude
32   AM2_COMMON_LATLON ll89ar[LMT][AM2_DEF_SNUM_LO]; // latlon for 89a altitude
33   AM2_COMMON_LATLON ll89br[LMT][AM2_DEF_SNUM_LO]; // latlon for 89b altitude
34
35   float tb06h06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06h, resolution 06G
36   float tb06v06 [LMT][AM2_DEF_SNUM_LO]; // tb for 06v, resolution 06G
37
38   unsigned char pdq06h [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06h
39   unsigned char pdq06v [LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06v
40
41   unsigned char lof06 [LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 06
42   unsigned char lof10 [LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 10
43
44   float ear_in[AM2_DEF_SNUM_LO][LMT]; // earth incidence
45   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.

Define the variables for datasets.

Limit of scan number is already defined 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.

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

- Open the HDF5 file.
  ```c
  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.
  ```

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

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

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

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.

```
if(ret<0){
  printf("AMTK_getScanTime error.\n");
  printf("amtk status: %d\n",ret);
  exit(1);
}
printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d\n",
  st[0].year,
  st[0].month,
  st[0].day,
  st[0].hour,
  st[0].minute,
  st[0].second);
```

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

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.

```
if(ret<0){
  printf("AMTK_getLatLon error: AM2_LATLON_RS_89A\n");
  printf("amtk status: %d\n",ret);
  exit(1);
}
printf("latlon89ar[scan=0][pixel=0]: (%9.4f,%9.4f)\n", l189ar[0][0].lat, l189ar[0][0].lon);
```
Read brightness temperature

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

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.

```c
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]);
```

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

```
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   }
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 array: land ocean flag for low
vpnt=loflo;
ret=AMTK_get_SwathInt(hnd,(int **)vpnt,1,num,AM2_LOF_RES_LO);
if(ret<0){
  printf("AMTK_get_SwathInt error: AM2_LOF_RES_LO\n");
  printf("amtk status: %d\n",ret);
  exit(1);
}
```

“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.

```
for(j=0;j<num;++j){
  for(i=0;i<AM2_DEF_SNUM_LO;++i){
    lof06[j][i]=loflo[num*0+j][i];
    lof10[j][i]=loflo[num*1+j][i];
    lof23[j][i]=loflo[num*2+j][i];
    lof36[j][i]=loflo[num*3+j][i];
  }
}
```

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

```
// read array: earth incidence
vpnt=ear_in;
ret=AMTK_get_SwathFloat(hnd,(float **)vpnt,1,num,AM2_EARTH_INC);
if(ret<0){
  printf("AMTK_get_SwathFloat error: AM2_EARTH_INC\n");
  printf("amtk status: %d\n",ret);
  exit(1);
}
```

Earth incidence data is two dimensional (sample * scan), and data from 1 scan to “num” scan is read in the sample program.

```
for(j=0;j<num;++j){
  for(i=0;i<AM2_DEF_SNUM_LO;++i){
    ear_in[j][i]=ear_in[num*0+j][i];
  }
}
```

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

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

hnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.
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 >
$ 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 readL1R_amtk_c” is shown as follows.

```
$ ./readL1R_amtk_c GW1AM2_201207261145_055A_L1S
GRTBR_0000000.h5

GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1S GRTBR_00

ObservationStartDateTime: 2012-07-26T11:45:43.0
EquatorCrossingDateTime: 2012-07-26T12:37:84.8Z
ObservationEndDateTime: 2012-07-26T12:35:09.735Z
NumberOfScans: 1979
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

 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)
latlon89lr[scan=0][pixel=0]: ( -73.3581, 136.8432)

pdq06h[scan=0][pixel=0]: 173.41
pdq06v[scan=0][pixel=0]: 173.41
pdq07h[scan=0][pixel=0]: 173.07
pdq07v[scan=0][pixel=0]: 173.07
pdq10h[scan=0][pixel=0]: 207.11
pdq10v[scan=0][pixel=0]: 207.11
pdq18h[scan=0][pixel=0]: 170.40
pdq18v[scan=0][pixel=0]: 170.40
pdq23h[scan=0][pixel=0]: 165.83
pdq23v[scan=0][pixel=0]: 165.83
pdq36h[scan=0][pixel=0]: 183.95
pdq36v[scan=0][pixel=0]: 183.95
pdq89ah[scan=0][pixel=0]: 163.86
pdq89av[scan=0][pixel=0]: 163.86
pdq89bh[scan=0][pixel=0]: 181.05
pdq89bv[scan=0][pixel=0]: 181.05

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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>* Longitude of Observation Point</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>* Geophysical Data</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>* Pixel Data Quality</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td></td>
</tr>
<tr>
<td>- OverlapScans</td>
<td></td>
</tr>
</tbody>
</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 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
8
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
18   // meta data
19   char geo[512]; // GeophysicalName
20   char gid[512]; // GranuleID
21   char tm1[512]; // ObservationStartDateTime
22   char tm2[512]; // EquatorCrossingDateTime
23   char tm3[512]; // ObservationEndDateTime
24   int num;        // NumberOfScans
25   int ovr;        // OverlapScans
26
27   // array data
28   AM2_COMMON_SCANTIME st[LMT]; // scantime
29   AM2_COMMON_LATLON ll[LMT][AM2_DEF_SNUM_LO]; // latlon
30   float geo1[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 1
31   float geo2[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 2
32   unsigned char pdq1[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 1
33   unsigned char pdq2[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 2
34   unsigned char pdqtmp[LMT*2][AM2_DEF_SNUM_LO]; // pixel data quality temporary
35
36 // include header file of AMTK for C language.
37 Define interface variables for AMTK.
38 Define the variables for metadata.
39 Define the variables for datasets.
40
41 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.
42 Limit of scan number is already defined in this program, because it also varies with products. (LMT=2200)
43 "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.
44 "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.
Open HDF file * Numbers written on the left are row number of the sample program.

```c
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.

```c
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);
```

```c
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);
```

Open the HDF5 file.

**hnd**: HDF access file id. Normal: HDF access file id is returned. Error: A negative value is returned.

**fn**: AMSR2 HDF file name.

Read metadata.

**hnd**: HDF access file id.

**met**: Metadata name.

**out**: Metadata value.

**ret**: Normal: The number of meta data character is returned. Error: A negative value is returned.

**AMTK_getMetaDataName(hnd,met,out)**

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

**AMTK_openH5(fn)**

`fn`: AMSR2 HDF file name.

~Number of scan is necessary when reading datasets.~

**AMTK_getMetaDataName(hnd,met,out)**

~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.

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.
```

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

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.
```
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.

You can read the second layer by changing the access label.
Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

```c
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.

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
```

```c
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
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.
Close the HDF5 file.

```c
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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc=readL2L_amtk.c

# output filename
out=readL2L_amtk_c

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
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"

echo $cmd
$cmd

# garbage
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 >
$ 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 “readL2L_amtk_c” is shown as follows.

```
$ ./readL2L_amtk_c GW1AM2_201303011809_125D_L2SGCLWLAL0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLAL0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_201303011809_125D_L2SGCLWLAL0000000
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDate</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDate</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Geophysical Data for 89A</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Geophysical Data for 89B</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality for 89A</td>
</tr>
<tr>
<td></td>
<td>- Pixel Data Quality for 89B</td>
</tr>
</tbody>
</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 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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "AMTK.h"

// fixed value
#define LMT 2200 // limit of NumberOfScans

int main(int argc, char *argv[]){
    // interface variable
    int i,j;       // loop variable
    int ret;       // return status
    char buf[512]; // text buffer
    void *vpnt;    // pointer to void
    char *fn;      // filename
    hid_t hnd;     // file handle

    // meta data
    char geo[512];  // GeophysicalName
    char gid[512];  // GranuleID
    char tm1[512];  // ObservationStartDateTime
    char tm2[512];  // EquatorCrossingDateTime
    char tm3[512];  // ObservationEndDateTime
    int num;        // NumberOfScans
    int ovr;        // OverlapScans

    // array data
    AM2_COMMON_SCANTIME st[LMT]; // scantime
    AM2_COMMON_LATLON ll89a[LMT][AM2_DEF_SNUM_HI];
    AM2_COMMON_LATLON ll89b[LMT][AM2_DEF_SNUM_HI];
    float geo1_89a[LMT][AM2_DEF_SNUM_HI];
    float geo1_89b[LMT][AM2_DEF_SNUM_HI];
    unsigned char pdq1_89a[LMT][AM2_DEF_SNUM_HI];
    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.

Define the variables for datasets.

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

```
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.

```
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);
```

```
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);
```
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.

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", st[0].year,
151       st[0].month,
152       st[0].day,
153       st[0].hour,
154       st[0].minute,
155       st[0].second
156   );

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

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
162     ,AM2_LATLON_L2_89A);
163   if(ret<0){
164     printf("AMTK_getLatLon error: AM2_LATLON_L2_89A¥n");
165     printf("amtk status: %d¥n",ret);
166     exit(1);
167   }
168   printf("latlon89a[scan=0][pixel=0]: (%9.4f,%9.4f)¥n", ll89a[0][0].lat,
169     ll89a[0][0].lon);
Read geophysical quantities * Numbers written on the left are row number of the sample program.

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.

```c
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.
```

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

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.

```c
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.
```

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.

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

Close the HDF5 file.

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

```bash
#!/bin/sh
#
### environment
export LANG=C
#

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc=readL2H_amtk.c

# output filename
out=readL2H_amtk_c

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
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"
echo $cmd
$cmd

# garbage
rm -f *.o
```

The execution example of “build_readL2H_amtk_c.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./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 >
$ 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 “readL2H_amtk_c” is shown as follows.

```
$ ./readL2H_amtk_c GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHA0000000.h5
GranuleID: GW1AM2_201303011809_125D_L2SGPRCHA0000000
GeophysicalName: Precipitation
ObservationStartDate: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDate: 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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Brightness Temperature (H)</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>- Brightness Temperature (V)</td>
</tr>
</tbody>
</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, 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.
AMSR2 data users manual

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "AMTK.h"

int main(int argc, char *argv[]){
  // interface variable
  int i,j;       // loop variable
  int ret;       // return status
  char buf[512]; // text buffer
  void *vpnt;    // pointer to void
  char *fn;      // filename
  hid_t hnd;     // file handle
  int siz[3];    // array size
  int x;         // grid size x
  int y;         // grid size y
  // meta data
  char geo[512];  // GeophysicalName
  char gid[512];  // GranuleID
  // array data
  float *tbH; // brightness temperature for horizontal
  float *tbV; // brightness temperature for vertical

  // Open HDF 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.

  // memory allocation
  Allocating memory will be held after researching the size of datasets.
```

Include header file of AMTK for C language.

Define interface variables for AMTK.

Define the variables for metadata.

Define the variables for datasets.

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

```c
45   // open
46   hnd=AMTK_openH5(fn);
47   if(hnd<0){
48     printf("AMTK_openH5 error: %s", fn);
49     printf("amtk status: %d", hnd);
50     exit(1);
51   }
```
Read metadata * Numbers written on the left are row number of the sample program.

```c
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.
```

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

Allocate the memory.

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

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

```c
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.
```

Allocate the memory.

```c
// memory allocate
98   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.

```c
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.
```

```c
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.

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

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

```c
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.

```bash
#!/bin/sh
3 ### environment
4 export LANG=C

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

11 # compiler
12 cc=icc

14 # source filename
15 csrc=readL3B_amtk.c

17 # output filename
18 out=readL3B_amtk_c

20 # library order
21 lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

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
28 # garbage
29 rm -f *.o
```

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

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

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 >
$ 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 “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)
+------------------------------------------------------------------------+
| 41123344444444444444444444444444444444444444444444433443222222444444 |
| 111134311113444444444444444444444444444444444444444444444444233334311 |
| 12452345555555444444444444444444444444444444444444444145554444444442112111111 |
| 154544444444444444444444444544111111111111111113554444444444222111111112 |
| 41224254414444444444445341131111111111111111114555444455111111111115 |
| 4444444444445555541111111111111111111111111111111115555553911111111111124 |
| 44444444444444411555155531111111111111111111111111111111155111111111111 |
| 55555555555511111211131112111111111111111111111111111111111111111111111144 |
| 115555555511111111111111111111111111111111111111111111111111111111111111 |
| 111155553111111111111111111111111111111111111111111111111111111111111111 |
| 111555552151111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 111111111111111111111111111111111111111111111111111111111111111111111111 |
| 34332344443332423433434444444434343321111111111111111111134444444111133 |
| 333333333333333333333333333333333333333333333333333333333333333 |
+------------------------------------------------------------------------+
```
### ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)

```
| 43334444444445545544444444444444444445444444444444444444433244544 |
| 333454334554544444444444444444444544444444444445544444444445444433 |
| 35534555555555555455554555455554555554555555555555555555555455544 |
| 355555555555555555555555555555555555555555555555555555555555555555 |
| 555555555555555555555555555555555555555555555555555555555555555555 |
| 655566655553333333333333333333333333333333333333333333333333333333 |
| 333555556333333333333333333333333333333333333333333333333333333333 |
| 333333333333333333333333333333333333333333333333333333333333333333 |
| 44433444444444444444444444444444444444444444444444444444444444444 |
| #]: missing  
 [ ]: out of observation  
 [1]: 50-100K  
 [2]: 100-150K  
 [3]: 150-200K  
 [4]: 200-250K  
 [5]: 250-300K  
 [6]: 300-350K  
 [*]: other  
```

---

AMS R2 data users manual
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Geophysical Data</td>
</tr>
<tr>
<td>- GranuleID</td>
<td></td>
</tr>
</tbody>
</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, 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.
AMSР2 data users manual

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "AMTK.h"

int main(int argc, char *argv[]){
  // interface variable
  int i,j;       // loop variable
  int ret;       // return status
  char buf[512]; // text buffer
  void *vpnt;    // pointer to void
  char *fn;      // filename
  hid_t hnd;     // file handle
  int siz[3];    // array size
  int x;         // grid size x
  int y;         // grid size y
  // meta data
  char geo[512];  // GeophysicalName
  char gid[512];  // GranuleID
  // array data
  float *geo1;   // geophysical data layer 1
  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.

```c
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.

// open
hnd=AMTK_openH5(fn);
if(hnd<0){
  printf("AMTK_openH5 error: %s", fn);
  printf("amtk status: %d", hnd);
  exit(1);
}
```

Open the HDF5 file.
Read metadata * Numbers written on the left are row number of the sample program.

```
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.

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 meta: GeophysicalName
vpnt=geo;
ret=AMTK_getMetaDataName(hnd,"GeophysicalName",(char **)&vpnt);
if(ret<0){
  printf("AMTK_getMetaDataName error: GeophysicalName¥n");
  printf("amtk status: %d¥n",ret);
  exit(1);
}
printf("GeophysicalName: %s¥n",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.

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.

```
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(strcmp(gid+29,"SND",3)==0 || strcmp(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(strcmp(strcmp(gid+29,"SND",3)==0 || strcmp(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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc=readL3G_amtk.c

# output filename
out=readL3G_amtk_c

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
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"

echo $cmd
$cmd

# garbage
rm -f *.o
```

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

```bash
$ ./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 >
$ 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 “readL3G_amtk_c” is shown as follows.

```
$ ./readL3G_amtk_c GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000
grid size x: 1440
grid size y: 720
```

<table>
<thead>
<tr>
<th>#</th>
<th>missing</th>
<th>[ ]: out of observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000 - 0.030 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.030 - 0.060 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.060 - 0.090 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.090 - 0.120 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.120 - 0.150 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.150 - 0.180 Kg/m²</td>
<td></td>
</tr>
<tr>
<td>[*]</td>
<td>other</td>
<td></td>
</tr>
</tbody>
</table>
6. FORTRAN90 Programming with AMTK

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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>- * Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDate-time</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDate-time</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDate-time</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (7.3GHz,V)</td>
</tr>
<tr>
<td>* Number of Scans</td>
<td>- Brightness Temperature (10.7GHz,H)</td>
</tr>
<tr>
<td>Data calculated from observation point of</td>
<td>- Brightness Temperature (10.7GHz,V)</td>
</tr>
<tr>
<td>89GHz-A</td>
<td>- Brightness Temperature (18.7GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (Low mean)</td>
<td>- Brightness Temperature (18.7GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (6G)</td>
<td>- Brightness Temperature (23.8GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (7G)</td>
<td>- Brightness Temperature (23.8GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (10G)</td>
<td>- Brightness Temperature (36.5GHz, H)</td>
</tr>
<tr>
<td>- Latitude and longitude (18G)</td>
<td>- Brightness Temperature (36.5GHz, V)</td>
</tr>
<tr>
<td>- Latitude and longitude (23G)</td>
<td>- Brightness Temperature (89.0GHz-A, H)</td>
</tr>
<tr>
<td>- Latitude and longitude (36G)</td>
<td>- Brightness Temperature (89.0GHz-A, V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-B, H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-B, V)</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality 6 to 36</td>
</tr>
<tr>
<td></td>
<td>- Pixel Data Quality 89</td>
</tr>
<tr>
<td></td>
<td>* Land_Ocean Flag 6 to 36</td>
</tr>
<tr>
<td></td>
<td>- Land_Ocean Flag 89</td>
</tr>
<tr>
<td></td>
<td>* Earth Incidence</td>
</tr>
<tr>
<td></td>
<td>- Earth Azimuth</td>
</tr>
</tbody>
</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.
AMS R2 data users manual

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) prml ! CoRegistrationParameterA1
24    character(len=512) prml ! 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
29
30     real(4) tb06h(AM2_DEF_SNUM_LO,LMT) ! tb for 06h
31     real(4) tb06v(AM2_DEF_SNUM_LO,LMT) ! tb for 06v

36     integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT) ! pixel data quality for 06h
37     integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT) ! pixel data quality for 06v

53     integer(1) lof06(AM2_DEF_SNUM_LO,LMT) ! land ocean flag for 06
54     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.

```c
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,a)')'amtk status: ',hnd
87      call exit(1)
88    endif
```

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

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

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

```c
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.
```

For details to P.14.

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

```c
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.
```

For details to P.20.
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, i, 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.

```fortran
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
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 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.
```

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

```
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
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
```

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
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.

```
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.

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

ret=AMTK_closeH5(hnd)

hnd: HDF access file id.

ret: [Return value] Error: A negative value is returned.
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.

```bash
#!/bin/sh

# environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort

# source filename
fsrc=readL1B_amtk.f

# output filename
out=readL1B_amtk_f

# library order
lib=-lAMSR2 -lhdf5 -lsz -lz -lm

# compile
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"
echo $cmd
$cmd

# garbage
rm -f *.o
```

The execution example of “build_readL1B_amtk_f.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./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 >
$ 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_f” is shown as follows.

```c
$ ./readL1B_amtk_f GW1AM2_201207261145_055A_L1S
GETBTR_0000000.h5
input file: GW1AM2_201207261145_055A_L1SGETBTR_0
0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGETBTR_0
000000
ObservationStartDate: 2012-07-26T11:45:43.0
18Z
EquatorCrossingDateTime: 2012-07-26T12:37:84
8Z
ObservationEndDate: 2012-07-26T12:35:09.735
Z
NumberOfScans: 1979
limit of NumberOfScans = 2200
OverlapScans: 20
CoRegistrationParameterA1: 6G-1.25000, 7G-1.0000
10G-1.25000, 18G-1.25000, 23G-1.0000
36G-1.0000
CoRegistrationParameterA2: 6G-0.00000, 7G--0.1000
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): (-73.3497, 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
```
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>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>- Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (res06,6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (res06,6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (res06,7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (res06,7.3GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,V)</td>
</tr>
<tr>
<td>* Pixel Data Quality 6 to 36</td>
<td>* Land_Ocean Flag 6 to 36</td>
</tr>
<tr>
<td>- Pixel Data Quality 89</td>
<td>- Land_Ocean Flag 89</td>
</tr>
<tr>
<td></td>
<td>* Earth Incidence</td>
</tr>
<tr>
<td></td>
<td>- Earth Azimuth</td>
</tr>
</tbody>
</table>

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 real(4) -> AMTK_get_SwathFloat function
AMSR2 data users manual

- 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.

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.

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.

```fortran
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) ! scante
27    type(AM2_COMMON_LATLON) l89ar(AM2_DEF_SNUM_HI,LMT) ! latlon for 89a
28    type(AM2_COMMON_SCANTIME) st(LMT) ! scante
29    real(4) tb06h06(AM2_DEF_SNUM_LO,LMT) ! tb for 06h, resolution 06G
30    real(4) tb06v06(AM2_DEF_SNUM_LO,LMT) ! tb for 06v, resolution 06G
31    integer(1) pdq06h(AM2_DEF_SNUM_LO,LMT) ! pixel data quality for 06h
32    integer(1) pdq06v(AM2_DEF_SNUM_LO,LMT) ! pixel data quality for 06v
33    integer(1) lof06(AM2_DEF_SNUM_LO,LMT) ! land ocean flag for 06
34    integer(1) lof10(AM2_DEF_SNUM_LO,LMT) ! land ocean flag for 10
35    real(4) ear_in(AM2_DEF_SNUM_LO,LMT) ! earth incidence
36    real(4) ear_az(AM2_DEF_SNUM_LO,LMT) ! earth azimuth
```
Open HDF file * Numbers written on the left are row number of the sample program.

```c
Open the HDF5 file.

```

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.

```c
Read the metadata.
```

83 C read meta: GeophysicalName
84   ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
85   if(ret.lt.0) then
86     write(*,'(a,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))

92 C read meta: NumberOfScans
93   ret=AMTK_getMetaDataName(hnd,'NumberOfScans',buf)
94   if(ret.lt.0) then
95     write(*,'(a,a)') 'AMTK_getMetaDataName error: NumberOfScans'
96     write(*,'(a,i12)') 'amtk status: ',ret
97     call exit(1)
98     endif
99     read(buf(1:ret),*) num
100    write(*,'(a,a)') '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.

```
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.
```

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

For details to P.14.
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.

```c
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.

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,pdqlo,1,num,AM2_PIX_QUAL_LO)
  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.
AMSR2 data users manual

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

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.

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.

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

ret=AMTK_closeH5(hnd)

hnd: HDF access file id.

ret: [Return value] Error: A negative value is returned.

For details to P18.
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.

```bash
1 #!/bin/sh
2
3 ### environment
4 export LANG=C
5
6 # library directory
7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
9 ZIP=/home/user1/util/szip_2.1
10
11 # compiler
12 fc=ifort
13
14 # source filename
15 src=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 $src -o $out -IAMTK/include -I$HDF5/include -I$ZIP/include
25 -LAMTK/lib -L$HDF5/lib -L$ZIP/lib $lib"
26 echo $cmd
27 $cmd
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.

```bash
$ ./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
-I/home/user1/util/AMTK_AMSR2_1.11/lib
-I/home/user1/util/hdf5_1.8.4-patch1/lib
-I/home/user1/util/szip_2.1/lib
-lAMSR2 -lhdf5 -lsz -lz -lm
```

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

Specify the compiler you use in row number 12. Intel compiler (ifort) or PGI compiler (pgf90) are required.
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 >  
$ 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 “readL1R_amtk_f” is shown as follows.

```
$ ./readL1R_amtk_f GW1AM2_201207261145_055A_LISGRTB_0000000.h5
input file: GW1AM2_201207261145_055A_LISGRTB_0000000.h5
GeophysicalName: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_LISGRTB_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, 18G-0.00000, 23G-0.00000, 36G-0.00000
CoRegistrationParameterA2: 6G-0.00000, 7G-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, 136.8432)
latlon89br(pixel=1, scan=1): (-73.4328, 137.2216)
latlon1r(pixel=1, scan=1): (-73.3581, 136.8432)
tb06h06(pixel=1, scan=1): 208.09
tb06v06(pixel=1, scan=1): 173.41
tb07h06(pixel=1, scan=1): 173.07
tb07v06(pixel=1, scan=1): 207.11
tb10h10(pixel=1, scan=1): 204.58
tb10v10(pixel=1, scan=1): 165.83
tb18h23(pixel=1, scan=1): 199.41
tb23h23(pixel=1, scan=1): 163.55
tb23v23(pixel=1, scan=1): 181.05
tb36h36(pixel=1, scan=1): 153.55
tb36v36(pixel=1, scan=1): 183.95
tb89h36(pixel=1, scan=1): 183.95
tb89v36(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) which reads the metadata and the datasets of L1B is shown below. Values of data are dumped to the standard output.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>* Longitude of Observation Point</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>* Geophysical Data</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>* Pixel Data Quality</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td></td>
</tr>
<tr>
<td>- OverlapScans</td>
<td></td>
</tr>
</tbody>
</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(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.*

```plaintext
1       program main
2       implicit none
3       include 'AMTK_f.h'

7       integer(4), parameter::LMT=2200 ! limit of NumberOfScans

12      integer(4) i,j         ! loop variable
13      integer(4) ret         ! return status
16      character(len=512) fn  ! filename
17      integer(4) hnd         ! file handle
19      character(len=512) geo  ! GeophysicalName
20      character(len=512) gid  ! GranuleID
22      character(len=512) tm1  ! ObservationStartDateTime
23      character(len=512) tm2  ! EquatorCrossingDateTime
24      integer(4) num          ! NumberOfScans
25      integer(4) ovr          ! OverlapScans

26      type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
27      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.
- Define the variables for datasets.

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

```c
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,a)')'amtk status: ',hnd
47     call exit(1)
48   endif
```

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

```c
49 C read meta: GeophysicalName
50   ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
51   if(ret.lt.0)then
52     write(*,'(a,a)')'AMTK_getMetaDataName error: GeophysicalName'
53     write(*,'(a,a)')'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.

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

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.

```fortran
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 
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.

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.

```fortran
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 
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.

```fortran
ret=AMTK_get_SwathFloat(hnd,out,bgn,end,label)
endf: 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.

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

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

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

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.

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           pdql1(i,j)=pdqtmp(i,num*0+j)
212           pdq2(i,j)=pdqtmp(i,num*1+j)
213           if(pdql1(i,j).ge.0)then
214             pdql1(i,j)=pdql1(i,j)
215           else
216             pdql1(i,j)=pdql1(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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort

# source filename
fsrc=readL2L_amtk.f

# output filename
out=readL2L_amtk_f

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
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"
echo $cmd
$cmd

# garbage
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 >
$ 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 “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
OverlapOfScans: 0
time(scan=1): 2013/03/01 18:09:10
latlon(pixel=1,scan=1): ( 84.4574, -78.1076)
geol(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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDate</td>
<td>* Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>* Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDate</td>
<td>* Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Geophysical Data for 89A</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>* Geophysical Data for 89B</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality for 89A</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality for 89B</td>
</tr>
</tbody>
</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(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.

```c
1  program main
2    implicit none
3  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(4) pdqtmp(AM2_DEF_SNUM_HI,LMT)
```

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.

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

```
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.

```
49 C read meta: GeophysicalName
50   ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
51   if(ret.lt.0)then
52     write(*,'(a,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,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.

```fortran
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.
```

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

```fortran
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.
```

...
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.

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.

```fortran
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
```

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](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.

```fortran
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.

```sh
#!/bin/sh

### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort

# source filename
fsrc=readL2H_amtk.f

# output filename
out=readL2H_amtk_f

# library order
lib="-lAMSR2 -lhf5 -lsz -lz -lm"

# compile
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"
echo $cmd
$cmd
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
-I/home/user1/util/AMTK_AMSR2_1.11/lib
-I/home/user1/util/hdf5_1.8.4-patch1/lib
-I/home/user1/util/szip_2.1/lib
-lAMSR2 -lhf5 -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 >
$ 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 “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)
latron89b(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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Brightness Temperature (H)</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Brightness Temperature (V)</td>
</tr>
</tbody>
</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, 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.
AMSAR2 data users manual

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(:,:)

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

Open the HDF5 file.

hnd=AMTK_openH5(fn)
fnd: 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.le.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
AMSR2 data users manual

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

```
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))
```

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

```
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.

```plaintext
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.

```plaintext
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
6 # library directory
7 AMTK=/home/user1/util/AMTK_AMSR2_1.11
8 HDF5=/home/user1/util/hdf5_1.8.4-patch1
9 ZIP=/home/user1/util/szip_2.1
10
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$HDF5/include -I$ZIP/include
26 echo $cmd
27 $cmd
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 >
$ 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 “readL3B_amtk_f” is shown as follows.

```bash
$ ./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)
+------------------------------------------------------------------------+  
| 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111}
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Geophysical Data</td>
</tr>
<tr>
<td>- GranuleID</td>
<td></td>
</tr>
</tbody>
</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, 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.
AMSRR2 data users manual

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(:,:)

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

Open the HDF5 file.

hlen=AMTK_openH5(fn)
fn: AMSRR2 HDF file name.

hlen: [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
AMSR2 data users manual

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

Read the metadata.

```fortran
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.
```

```fortran
ret=AMTK_getMetaDataName(hnd,'GeophysicalName',geo)
  if(ret.lt.0)then
    write(*,'(a)')'AMTK_getMetaDataName error: GeophysicalName'
    write(*,'(a,i12)')'amtk status: ',ret
    call exit(1)
  endif
  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.

```fortran
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.
```

```fortran
ret=AMTK_getDimSize(hnd,AM2_GRID_GEO1,siz);
  if(ret.lt.0)then
    write(*,'(a)')'AMTK_getDimSize error: AM2_GRID_GEO1'
    write(*,'(a,i12)')'amtk status: ',ret
    call exit(1)
  endif
  x=siz(2);
  y=siz(1);
  write(*,'(a,i12)')'grid size x: ',x
  write(*,'(a,i12)')'grid size y: ',y
```

Allocate the memory.

```fortran
allocate(geo1(x,y),stat=ret)
  if(ret.ne.0)then
    write(*,'(a)')'memory allocate error: geo1'
    call exit(1)
  endif
```
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”.

```
ret=AMTK_get_GridFloat(hnd,out,label)
```

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

Close HDF file

```
ret=AMTK_closeH5(hnd)
```

Release the memory.
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.

```bash
#!/bin/sh
### environment
export LANG=C

# library directory
AMTK=/home/user1/util/AMTK_AMSR2_1.11
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort

# source filename
fsrc=readL3G_amtk.f

# output filename
out=readL3G_amtk_f

# library order
lib="-lAMSR2 -lhdf5 -lsz -lz -lm"

# compile
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"
echo $cmd
$cmd

# garbage
rm -f *.o
```

The execution example of “build_readL3G_amtk_f.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./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
```
AMSR2 data users manual

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 >
$ 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 “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#############################00101112413333233###############343413131#|
|#20#######2###############00#1223452344213211120##########3355421131121#|
|#########################4#3222223232100110111110#######14*41111201111##|
|########0###000###0####111111112101201110*131121100##1#11#01211110100###|
|###########1011111111#211#11111210011121333223224321000#1###1111000000##|
|11#######00010122125#3###1#12232321100010020110001011122#######222321111|
|000######122131112311112#01112#11312212323110100000000110########1001110|
|100####32#12111141111000######2232322421212441331110011010######11111100|
|1000###1211232211251100########131112232110121313122011010#####231111001|
|11131212231110111111211111110012102#1231221223311213111021##011124223242|
|*33222511232222222223223331421022422322211024222222122113##0111421221132|
|101122112221111222111111111222312232232331243321212323221223312232222111|
|####################################311122122221112111221########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

130


7. C Programming with HDF5 library (without AMTK)

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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (7.3GHz,V)</td>
</tr>
<tr>
<td>Data calculated from observation point of 89GHz-A</td>
<td>- Brightness Temperature (10.7GHz,H)</td>
</tr>
<tr>
<td>* Latitude and longitude (Low mean)</td>
<td>- Brightness Temperature (10.7GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (6G)</td>
<td>- Brightness Temperature (18.7GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (7G)</td>
<td>- Brightness Temperature (18.7GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (10G)</td>
<td>- Brightness Temperature (23.8GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (18G)</td>
<td>- Brightness Temperature (23.8GHz,V)</td>
</tr>
<tr>
<td>- Latitude and longitude (23G)</td>
<td>- Brightness Temperature (36.5GHz,H)</td>
</tr>
<tr>
<td>- Latitude and longitude (36G)</td>
<td>- Brightness Temperature (36.5GHz,V)</td>
</tr>
<tr>
<td>For details to P.20</td>
<td></td>
</tr>
</tbody>
</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"
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; // EquatorCrossingDateTime
29   char *tm3 = NULL; // ObservationEndDateTime
30   int num;          // NumberOfScans
31   int ovr;          // OverlapScans
32   double prml[7];   // CoRegistrationParameterA1
33   double prm2[7];   // CoRegistrationParameterA2
34
35   // array data
36   AM2_COMMON_SCANTIME st[LMT]; // scantime
37   float lat89a[LMT][AM2_DEF_SNUM_HI]; // lat for 89a
38   float lat89b[LMT][AM2_DEF_SNUM_HI]; // lat for 89b
39   float lon89a[LMT][AM2_DEF_SNUM_HI]; // lon for 89a
40   float lon89b[LMT][AM2_DEF_SNUM_HI]; // lon for 89b
41   float tb06h[LMT][AM2_DEF_SNUM_LO]; // tb for 06h
42   float tb06v[LMT][AM2_DEF_SNUM_LO]; // tb for 06v
43   unsigned char pdq06h[LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06h
44   unsigned char pdq06v[LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06v
45   unsigned char lof06[LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 06
46   unsigned char lof07[LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 07
47   float ear_in[LMT][AM2_DEF_SNUM_LO]; // earth incidence
48   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.
Define the variables for datasets.
Initialize the HDF5 library.
ret = H5open();
ret: [Return value] Error: A negative value is returned.

Open an existing HDF5 file.
fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.

Open the HDF5 file.
fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
fhnd < 0){
printf("H5Fopen error: %s", fn);
exit(1);
}
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. 
\[ \text{ahnd} = \text{H5Aopen(fhnd, nam, label);} \]
\[ \text{fhnd}: \text{Object id.} \]
\[ \text{nam}: \text{Name of attribute.} \]
\[ \text{label}: \text{Use H5P_DEFAULT.} \]
\[ \text{ahnd}: \text{[Return value]} \text{ Normal: Attribute id is returned. Error: A negative value is returned.} \]

Get an attribute data type. 
\[ \text{atyp} = \text{H5Aget_type(ahnd);} \]
\[ \text{ahnd}: \text{Attribute id.} \]
\[ \text{atyp}: \text{[Return value]} \text{ Normal: Datatype id is returned. Error: A negative value is returned.} \]

Read an attribute. 
\[ \text{ret} = \text{H5Aread(ahnd, otyp, buf);} \]
\[ \text{ahnd}: \text{Attribute id.} \]
\[ \text{otyp}: \text{Datatype id. (Attribute is automatically converted to specified data type.)} \]
\[ \text{buf}: \text{Buffer for data to be read.} \]
\[ \text{ret}: \text{[Return value]} \text{ Error: A negative value is returned.} \]

Close the attribute. 
\[ \text{ret} = \text{H5Aclose(ahnd);} \]
\[ \text{ahnd}: \text{Attribute id.} \]
\[ \text{ret}: \text{[Return value]} \text{ Error: A negative value is returned.} \]

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);

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);
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.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>265</td>
<td>dhnd = H5Dopen(fhnd, nam, label);</td>
</tr>
<tr>
<td>266</td>
<td>fhnd: HDF access file id.</td>
</tr>
<tr>
<td>267</td>
<td>nam: The name of the dataset to access.</td>
</tr>
<tr>
<td>268</td>
<td>label: Use H5P_DEFAULT.</td>
</tr>
<tr>
<td>269</td>
<td>dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

For details to P.13.

Read raw data from a dataset.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>270</td>
<td>ret = H5Dread(dhnd, otyp, label1, label2, label3, buf);</td>
</tr>
<tr>
<td>271</td>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>272</td>
<td>otyp: Datatype id. (Dataset is automatically converted to specified data type.)</td>
</tr>
<tr>
<td>273</td>
<td>label1, label2: Use when reading part of the array. Use H5S_ALL for both labels when reading entire array.</td>
</tr>
<tr>
<td>274</td>
<td>label3: Use H5P_DEFAULT.</td>
</tr>
<tr>
<td>275</td>
<td>buf: Variable storing acquired data.</td>
</tr>
<tr>
<td>276</td>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

Close the dataset.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>277</td>
<td>ret = H5Dclose(dhnd);</td>
</tr>
<tr>
<td>278</td>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>279</td>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

// read array: scantime
265  dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
266  ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
267  if(ret < 0){
268    printf("H5Dread error: Scan Time\n");
269    exit(1);
270  }
271  ret = H5Dclose(dhnd);
272  // cutoff overlap
273  for(j = 0; j < num; ++j){
274    r8d1[j] = r8d1[j + ovr];
275  }
276  for(j = num; j < LMT; ++j){
277    r8d1[j] = 0;
278  }
279  // convert
280  amsr2time_(&num, r8d1, st);
281  // sample display
282  printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d\n",
283       st[0].year,
284       st[0].month,
285       st[0].day,
286       st[0].hour,
287       st[0].minute,
288       st[0].second,
289    );

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.

Scanning time 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.
AMS2 data users manual

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.

```c
// read array: latlon for 89a
// read lat
dhnd = H5Dopen(fhnd, "Latitude of Observation Point for 89A", H5P_DEFAULT);
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat89a);
if(ret < 0){
    printf("H5Dread error: Latitude of Observation Point for 89A\n");
    exit(1);
}
ret = H5Dclose(dhnd);

// read lon
dhnd = H5Dopen(fhnd, "Longitude of Observation Point for 89A", H5P_DEFAULT);
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon89a);
if(ret < 0){
    printf("H5Dread error: Longitude of Observation Point for 89A\n");
    exit(1);
}
ret = H5Dclose(dhnd);

// cutoff overlap
for(j = 0; j < num; ++j){
    for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
        lat89a[j][i] = lat89a[j+ovr][i];
        lon89a[j][i] = lon89a[j+ovr][i];
    }
}

for(j = num; j < LMT; ++j){
    for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
        lat89a[j][i] = 0;
        lon89a[j][i] = 0;
    }
}
```

AMS2 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.

```c
// read array: latlon for low mean
amsr2latlon_(num, prm1[6], prm2[6], lat89a, lon89a, latlm, lonlm);
```

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.

```c
386   // read array: tb for 06h
387   dhnd = H5Dopen(fhnd, "Brightness Temperature (6.9GHz,H)", H5P_DEFAULT);
388   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT); // get scale
389   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);        // get scale
390   ret = H5Aclose(ahnd);                                    // get scale
391   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, tb06h);
392   if(ret < 0){ printf("H5Dread error: Brightness Temperature (6.9GHz,H)\n"); exit(1); }
393   ret = H5Dclose(dhnd);
394   // cutoff overlap & change scale
395   for(j = 0; j < num; ++j){
396     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
397       tb06h[j][i] = tb06h[j+ovr][i];
398       if(tb06h[j][i] < 65534) tb06h[j][i] = tb06h[j][i] * sca;
399     }
400   for(i = num; j < LMT; ++i){
401     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
402       tb06h[j][i] = 0;
403     }
404     printf("tb06h[scan=0][pixel=0]: %.2f\n", tb06h[0][0]);
410   // sample display
411   printf("tb06h[scan=0][pixel=0]: %9.2f\n", tb06h[0][0]);
```

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 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.

```c
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);

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     }
822   }
823   for(j = num; j < LMT; ++j){
824     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
825       pdq06h[j][i]=0;
826       pdq06v[j][i]=0;
827     }
828   }
```

Read L1B pixel data quality

Split the bit by bit information to each array of frequency and polarization.

Remove overlap scans also applied.
Read L1B land ocean flag

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).

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.

```
875 // read array: land ocean flag for low
876 dhnd = H5Dopen(fhnd, "Land_Ocean Flag 6 to 36", H5P_DEFAULT);
877 ret = H5Dread(dhnd, H5T_NATIVE_UCHAR, H5S_ALL, H5S_ALL, H5P_DEFAULT, loflo);
878 if(ret < 0){
879   printf("H5Dread error: Land_Ocean Flag 6 to 36\n");
880   exit(1);
881 }
882 ret = H5Dclose(dhnd);
883 
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]=loflo[(num+ovr*2)*0+j][i];
899     lof07[j][i]=loflo[(num+ovr*2)*1+j][i];
900     lof10[j][i]=loflo[(num+ovr*2)*2+j][i];
901     lof18[j][i]=loflo[(num+ovr*2)*3+j][i];
902     lof23[j][i]=loflo[(num+ovr*2)*4+j][i];
903     lof36[j][i]=loflo[(num+ovr*2)*5+j][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;
910     lof10[j][i]=0;
911     lof18[j][i]=0;
912     lof23[j][i]=0;
913     lof36[j][i]=0;
914   }
915 }
```

For details to P.18.
Read earth incidence * Numbers written on the left are row number of the sample program.

```
// read array: earth incidence
hnd = H5Open(fhnd, "Earth Incidence", H5P_DEFAULT);
and = H5Open(dhnd, "SCALE FACTOR", H5P_DEFAULT); // get scale
ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca); // get scale
ret = H5Aclose(ahnd); // get scale
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
ear_in);
if(ret < 0){
    printf("H5Dread error: Earth Incidence\n");
    exit(1);
}
ret = H5Dclose(dhnd);
// cutoff overlap & change scale
for(j = 0; j < num; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        ear_in[j][i] = ear_in[j+ovr][i];
        if(ear_in[j][i] > -32767) ear_in[j][i] = ear_in[j][i] * sca;
    }
}
for(j = num; j < LMT; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        ear_in[j][i] = 0;
    }
}
sample display
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
ret = H5Fcloss(fhnd);
ret = H5close();
```

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.

Close the HDF5 file.

```
ret = H5Fcloss(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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc="readL1B_hdf5.c amsr2time_.c amsr2latlon_.c"

# output filename
out=readL1B_hdf5_c

# library order
lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"

# c compile
$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib
$echo "$cmd"

# garbage
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
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-I/home/user1/util/szip_2.1/include
-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 >
$ 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_hdf5_c” is shown as follows.

```bash
$ ./readL1B_hdf5_c GW1AM2_201207261145_055A_L1SGBTBR_000000.h5
input file: GW1AM2_201207261145_055A_L1SGBTBR_000000.h5
Geophysical Name: Brightness Temperature
GranuleID: GW1AM2_201207261145_055A_L1SGBTBR_000000
Observation Start DateTime: 2012-07-26T11:45:43.018Z
Equator Crossing DateTime: 2012-07-26T12:12:37.848Z
Observation End DateTime: 2012-07-26T12:35:09.735Z
Number of Scans: 1979
limit of Number of Scans = 2200
Overlap Scans: 20
CoRegistration Parameter A1: 6G - 1.25000, 7G - 1.00000, 10G - 1.25000, 18G - 1.25000, 23G - 1.25000, 36G - 1.00000
CoRegistration Parameter A2: 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/comm on/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
time [scan=0]: 2012/07/26 11:45:43
lai08a[scan=0][pixel=0]: (-73.3289, 136.7714)
lai08b[scan=0][pixel=0]: (-73.4038, 137.1498)
lai08m[scan=0][pixel=0]: (-73.3538, 136.6228)
lai08n[scan=0][pixel=0]: (-73.3592, 136.6233)
lai08o[scan=0][pixel=0]: (-73.3497, 136.6429)
lai08p[scan=0][pixel=0]: (-73.3506, 136.6001)
lai08q[scan=0][pixel=0]: (-73.3592, 136.6213)
lai08r[scan=0][pixel=0]: (-73.3506, 136.6001)
lai08s[scan=0][pixel=0]: (-73.3532, 136.6514)
tb06h[scan=0][pixel=0]: 164.85
tb06v[scan=0][pixel=0]: 199.84
tb07h[scan=0][pixel=0]: 163.22
tb07v[scan=0][pixel=0]: 196.53
tb10h[scan=0][pixel=0]: 156.66
tb18h[scan=0][pixel=0]: 186.39
tb09a[scan=0][pixel=0]: 163.76
tb09v[scan=0][pixel=0]: 179.27
tb09h[scan=0][pixel=0]: 170.60
tb09v[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
lof89[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
```

limit of Number of scans = 2200
```
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>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>- Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Brightness Temperature (res06,6.9GHz,H)</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Brightness Temperature (res06,6.9GHz,V)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA1</td>
<td>- Brightness Temperature (res06,7.3GHz,H)</td>
</tr>
<tr>
<td>- CoRegistrationParameterA2</td>
<td>- Brightness Temperature (res06,7.3GHz,V)</td>
</tr>
</tbody>
</table>

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
AMSR2 data users manual

**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. // 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

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 prml[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
43
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
46
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
49
50   unsigned char pdq06h[LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06h
51   unsigned char pdq06v[LMT][AM2_DEF_SNUM_LO]; // pixel data quality for 06v
52
53   unsigned char lof06[LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 06
54   unsigned char lof10[LMT][AM2_DEF_SNUM_LO]; // land ocean flag for 10
55
56   float ear_in[LMT][AM2_DEF_SNUM_LO]; // earth incidence
57   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.
Define the variables for datasets.
Open HDF file * Numbers written on the left are row number of the sample program.

```c
// hdf5 initialize
ret = H5open();
if(ret < 0){
    printf("h5open error: %d\n",ret);
    exit(1);
}

// open
fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
if(fhnd < 0){
    printf("H5Fopen error: %s\n", fn);
    exit(1);
}
```

Initialize the HDF5 library.

Ret: [Return value] Error: A negative value is returned.

Open an existing HDF5 file.

Fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.
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.

```c
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.

```c
atyp = H5Aget_type(ahnd);
ahnd: Attribute id.
atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.
```

Read an attribute.

```c
ret = H5Aread(ahnd, otyp, buf);
ahnd: Attribute id.
atyp: 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.
```

Close the attribute.

```c
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.

```c
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);
```

```c
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); // All values of metadata are acquired as character type, so you need to convert the value to numerical.
173 printf("NumberOfScans: %d\n", num);
```
Read scanning time * Numbers written on the left are row number of the sample program.

```
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);
```

**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.
ootyp: 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.
```

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

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.

Read latitude.

```c
// read array: latlon for 89a altitude revised
// read lat
dhnd = H5Dopen(fhnd, "Latitude of Observation Point for 89A", H5P_DEFAULT);
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat89ar);
if(ret < 0){
  printf("H5Dread error: Latitude of Observation Point for 89A\n");
  exit(1);
}
ret = H5Dclose(dhnd);
// read lon
dhnd = H5Dopen(fhnd, "Longitude of Observation Point for 89A", H5P_DEFAULT);
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon89ar);
if(ret < 0){
  printf("H5Dread error: Longitude of Observation Point for 89A\n");
  exit(1);
}
ret = H5Dclose(dhnd);
// cutoff overlap
for(j = 0; j < num; ++j){
  for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
    lat89ar[j][i] = lat89ar[j+ovr][i];
    lon89ar[j][i] = lon89ar[j+ovr][i];
  }
}
for(j = num; j < LMT; ++j){
  for(i = 0; i < AM2_DEF_SNUM_HI; ++i){
    lat89ar[j][i] = 0;
    lon89ar[j][i] = 0;
  }
}
```

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.

```c
// read array: tb for 06h, resolution 06G
ahnd = H5Dopen(fhnd, "Brightness Temperature (res06, 6.9GHz, H)", H5P_DEFAULT);
ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca); // get scale
ret = H5Dread(dhnd,H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, tb06h06);
if(ret < 0){
    printf("H5Dread error: Brightness Temperature (res06, 6.9GHz, H)\n");
    exit(1);
}
ret = H5Dclose(dhnd);
// cutoff overlap & change scale
for(j = 0; j < num; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        tb06h06[j][i] = tb06h06[j+ovr][i];
        if(tb06h06[j][i] < 65534) tb06h06[j][i] = tb06h06[j][i] * sca;
    }
}
for(j = num; j < LMT; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        tb06h06[j][i] = 0;
    }
}
// sample display
printf("tb06h06[scan=0][pixel=0]: %9.2f\n", tb06h06[0][0]);
```

Read brightness temperature.  
Read scale factor.  
Remove overlap scans.  
Exclude the missing value (65535) and error value (65534) when scale handling is applied.
AMSR2 data users manual

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.

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.
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).

```
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);
```

“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   }
```

For details to P.18.

Read L1R land ocean flag.

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.

```
// read array: earth incidence
968   // read array: earth incidence
969   dhnd = H5Dopen(fhnd, "Earth Incidence", H5P_DEFAULT);
970   ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT); // get scale
971   ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);        // get scale
972   ret = H5Aclose(ahnd);                                    // get scale
973   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
974     ear_in);
975   if(ret < 0){
976     printf("H5Dread error: Earth Incidence\n");
977     exit(1);
978   }
979   ret = H5Dclose(dhnd);
980   // cutoff overlap & change scale
981   for(j = 0; j < num; ++j){
982     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
983       ear_in[j][i] = ear_in[j+ovr][i];
984       if(ear_in[j][i] > -32767) ear_in[j][i] = ear_in[j][i] * sca;
985     }
986   }
987   for(j = num; j < LMT; ++j){
988     for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
989       ear_in[j][i] = 0;
990     }
991   }
992   // sample display
993   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.

```
1020   // close
1021   ret = H5Fclose(fhnd);
1022   ret = H5close();
```

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.

Read scale factor.

Read earth incidence.

Remove overlap scans.

Exclude the missing value (-32768) and error value(-32767) when scale handling is applied.

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.

```bash
#!/bin/sh
#
### environment
export LANG=C
#
# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1
#
# compiler
cc=icc
#
# source filename
csrc="readL1R_hdf5.c amsr2time_.c "
#
# output filename
out=readL1R_hdf5_c
#
# library order
lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"
#
# c compile
# I use gnu compiler (gcc), Intel compiler (icc), or PGI compiler (pgcc) as required.
ccommand="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib"
echo $ccommand
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/hdf5_1.8.4-patch1/lib
-L/home/user1/util/szip_2.1/lib
-lhdf5_hl -lhdf5 -lsz -lz -lm
```

Specify the library directories in row number 7-8. “include” and “lib” directories are necessary under the each library directories.

Specify the compiler you use in row number 11. Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.
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 >
$ 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 “readL1R_hdf5_c” is shown as follows.

```
$ ./readL1R_hdf5_c G
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/comm
on/AMTK_AMSR2_DATA/leapsec.dat
amsr2time: year=1993 month= 7 tai93sec=   15638401.0
amsr2time: year=1994 month= 7 tai93sec=   47174402.0
amsr2time: year=1996 month= 1 tai93sec=   94608003.0
amsr2time: year=1997 month= 7 tai93sec=  141868804.0
amsr2time: year=1999 month= 1 tai93sec=  189302405.0
amsr2time: year=2006 month= 1 tai93sec=  410227206.0
amsr2time: year=2009 month= 1 tai93sec=  504921607.0
amsr2time: year=2012 month= 7 tai93sec=  615254408.0
amsr2time: number of leap second = 8
time[scan=0]: 2012/07/26 11:45:43
lation[8ar[scan=0][pixel=0]: (-73.3581, 136.8432)
lation8br[scan=0][pixel=0]: (-73.4328, 137.2216)
lationir[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
tb10h01[scan=0][pixel=0]: 170.40
tb10v01[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
```

```
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point</td>
</tr>
<tr>
<td>- ObservationStartDateDateTime</td>
<td>* Longitude of Observation Point</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>* Geophysical Data</td>
</tr>
<tr>
<td>- ObservationEndDateDateTime</td>
<td>* Pixel Data Quality</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td></td>
</tr>
<tr>
<td>- OverlapScans</td>
<td></td>
</tr>
</tbody>
</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.

```c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "hdf5.h"
5 #include "amsr2time.h"

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
15 int main(int argc, char *argv[]){
16   // interface variable
17   int i,j;          // loop variable
18   herr_t ret;       // return status
19   char *buf = NULL; // text buffer
20   char *fn = NULL; // filename
21   hid_t fhnd;       // file handle
22   hid_t ahnd;       // attribute handle
23   hid_t atyp;       // attribute type
24   hid_t dhnd;       // dataset handle
25     // meta data
26     char *geo = NULL; // GeophysicalName
27     char *gid = NULL; // GranuleID
28     char *tm1 = NULL; // ObservationStartDateTime
29     char *tm2 = NULL; // EquatorCrossingDateTime
30     char *tm3 = NULL; // ObservationEndDateTime
31     int num;          // NumberOfScans
32     int ovr;          // OverlapScans
33   37 AM2_COMMON_SCANTIME st[LMT]; // scantime
34   39 float lat[LMT][AM2_DEF_SNUM_LO]; // lat
40   40 float lon[LMT][AM2_DEF_SNUM_LO]; // lon
42   42 float geo1[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 1
43   43 float geo2[LMT][AM2_DEF_SNUM_LO]; // geophysical data layer 2
44   44 float geotmp[LMT*2][AM2_DEF_SNUM_LO]; // geophysical data temporary
45   46 unsigned char pdq1[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 1
46   48 unsigned char pdq2[LMT][AM2_DEF_SNUM_LO]; // pixel data quality layer 2
49   49 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.

Include header file of HDF5 for C language.

Define interface variables for HDF5.

Define the variables for metadata.

Define the variables for datasets.
Initialize the HDF5 library.

Open HDF file

```c
// hdf5 initialize
ret = H5open();
if(ret < 0){
    printf("h5open error: %d\n",ret);
    exit(1);
}
```

Open an existing HDF5 file.

```c
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.
```

```c
fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.
```

```c
// open
fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
if(fhnd < 0){
    printf("H5Fopen error: %s\n", fn);
    exit(1);
}
```
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.
```

---

```c
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.
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.

Read scanning time

// read array: scantime

// read array: scantime

// sample display

printf("time[scan=0]: %04d/%02d/%02d %02d:%02d:%02d\n", st[0].year, st[0].month, st[0].day, st[0].hour, st[0].minute, st[0].second);

Time conversion.

amsr2time_(num, in, out)
um: Number of scan.
in: Scanning time data of TAI93.
out: [Return value] Converted value is stored in AM2_COMMON_SCANTIME data structure.

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 see P.14.
Read latitude and longitude * Numbers written on the left are row number of the sample program.

```c
200   // read lat
201   dhnd = H5Dopen(fhnd, "Latitude of Observation Point", H5P_DEFAULT);
202   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lat);
203   if(ret < 0){
204     printf("H5Dread error: Latitude of Observation Point\n");
205     exit(1);
206   }
207   ret = H5Dclose(dhnd);
208   // read lon
209   dhnd = H5Dopen(fhnd, "Longitude of Observation Point", H5P_DEFAULT);
210   ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, lon);
211   if(ret < 0){
212     printf("H5Dread error: Longitude of Observation Point\n");
213     exit(1);
214   }
215   ret = H5Dclose(dhnd);
216   // sample display
217   printf("latlon[scan=0][pixel=0]: (%9.4f,%9.4f)\n", lat[0][0], lon[0][0]);
```
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.

Scale handling is automatically applied when using AMTK. For details to P.22.

Read geophysical quantities.

Read scale factor.

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.

// read array: geophysical data for 1 layer
if(strcmp(gid+29,"SND",3)!=0 || strcmp(gid+29,"SST",3)!=0){
dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);

// get scale
ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
ret = H5Aclose(ahnd);

// read
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, geo1);
if(ret < 0){
    printf("H5Dread error: Geophysical DataYn");
    exit(1);
}
ret = H5Dclose(dhnd);

// change scale
for(j = 0; j < num; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        if(geo1[j][i] > -32767) geo1[j][i] = geo1[j][i] * sca;
    }
}

// read array: geophysical data for 2 layer
if(strcmp(gid+29,"SND",3)==0 || strcmp(gid+29,"SST",3)==0){
dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);

// get scale
ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
ret = H5Aclose(ahnd);

// read
ret = H5Dread(dhnd, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, geotmp);
if(ret < 0){
    printf("H5Dread error: Geophysical DataYn");
    exit(1);
}
ret = H5Dclose(dhnd);

// separate & change scale
for(j = 0; j < num; ++j){
    for(i = 0; i < AM2_DEF_SNUM_LO; ++i){
        geo1[j][i] = geotmp[j*2][i*2+0];
        geo2[j][i] = geotmp[j*2][i*2+1];
        if(geo1[j][i] > -32767) geo1[j][i] = geo1[j][i] * sca;
        if(geo2[j][i] > -32767) geo2[j][i] = geo2[j][i] * sca;
    }
}
Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

```c
260   // read array: pixel data quality for 1 layer
261   if(strcmp(gid+29,"SND",3)!=0 && strcmp(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
271   // read array: pixel data quality for 2 layer
272   if(strcmp(gid+29,"SND",3)==0 || strcmp(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   }
```

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

```
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.

```c
314   // close
315   ret = H5Fclose(fhnd);
316   ret = H5close();
```

```
Close the HDF5 file.
Terminate access to an HDF5 file.
Flush all data to disk, close all open identifiers, and clean up memory.
```

163
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
6 # library directory
7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
8 SZIP=/home/user1/util/szip_2.1
9
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
-LSZIP/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:
  
  $ 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 “readL2L_hdf5_c” is shown as follows.

```
$ ./readL2L_hdf5_c GW1AM2_201303011809_125D_L2SGCLWLAL0000000.h5
input file: GW1AM2_201303011809_125D_L2SGCLWLAL0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_201303011809_125D_L2SGCLWLAL0000000
ObservationStartDate: 2013-03-01T18:09:10.122Z
EquatorCrossingDateTime: 2013-03-01T18:35:54.849Z
ObservationEndDate: 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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationStartDateTime</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>- EquatorCrossingDateTime</td>
<td>* Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>- ObservationEndDateTime</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* NumberOfScans</td>
<td>* Geophysical Data for 89A</td>
</tr>
<tr>
<td>- OverlapScans</td>
<td>- Geophysical Data for 89B</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality for 89A</td>
</tr>
<tr>
<td></td>
<td>- Pixel Data Quality for 89B</td>
</tr>
</tbody>
</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
AMSР2 data users manual

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hdf5.h"
#include "amsr2time.h"

// fixed value
#define LMT 2200 // limit of NumberOfScans
#define AM2_DEF_SNUM_HI 486 // high resolution pixel width
#define AM2_DEF_SNUM_LO 243 // low resolution pixel width

int main(int argc, char *argv[])
{
    // interface variable
    int i,j;          // loop variable
    herr_t ret;       // return status
    char *buf = NULL; // text buffer
    char *fn  = NULL; // filename
    hid_t fhnd;       // file handle
    hid_t ahnd;       // attribute handle
    hid_t atyp;       // attribute type
    hid_t dhnd;       // dataset handle

    // meta data
    char *geo = NULL; // GeophysicalName
    char *gid = NULL; // GranuleID
    char *tm1 = NULL; // ObservationStartDateTime
    char *tm2 = NULL; // EquatorCrossingDateTime
    char *tm3 = NULL; // ObservationEndDateTime
    int num;          // NumberOfScans
    int ovr;          // OverlapScans

    // array data
    AM2_COMMON_SCANTIME st[LMT]; // scantime
    float lat89a[LMT][AM2_DEF_SNUM_HI]; // lat for 89a
    float lat89b[LMT][AM2_DEF_SNUM_HI]; // lat for 89b
    float lon89a[LMT][AM2_DEF_SNUM_HI]; // lon for 89a
    float lon89b[LMT][AM2_DEF_SNUM_HI]; // lon for 89b
    float geol_89a[LMT][AM2_DEF_SNUM_HI];
    float geol_89b[LMT][AM2_DEF_SNUM_HI];
    unsigned char pdq1_89a[LMT][AM2_DEF_SNUM_HI];
    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.

```
58   // hdf5 initialize
59   ret = H5open();
60   if(ret < 0){
61     printf("h5open error: %d\n",ret);
62     exit(1);
63   }
```

Open an existing HDF5 file.

```
65   // open
66   fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
67   if(fhnd < 0){
68     printf("H5Fopen error: %s", fn);
69     exit(1);
70   }
```
Read metadata * Numbers written on the left are row number of the sample program.

<table>
<thead>
<tr>
<th>Open an attribute.</th>
<th>Get an attribute data type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ahnd = H5Aopen(fhnd, nam, label);</td>
<td>atyp = H5Aget_type(ahnd);</td>
</tr>
<tr>
<td>fhnd: Object id.</td>
<td>ahnd: Attribute id.</td>
</tr>
<tr>
<td>nam: Name of attribute.</td>
<td>atyp: [Return value] Normal: Datatype id is</td>
</tr>
<tr>
<td>label: Use H5P_DEFAULT.</td>
<td>returned. Error: A negative value is returned.</td>
</tr>
<tr>
<td>ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.</td>
<td></td>
</tr>
</tbody>
</table>

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.

```
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);
```

```
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.

```c
166   // read array: scantime
167   dhnd = H5Dopen(fhnd, "Scan Time", H5P_DEFAULT);
168   ret = H5Dread(dhnd, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL, H5P_DEFAULT, r8d1);
169   if(ret < 0){
170     printf("H5Dread error: Scan Time\n");
171     exit(1);
172   }
173   ret = H5Dclose(dhnd);
174   // convert
175   amsr2time_(&num, r8d1, st);
176   // 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 }
```

- 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.

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

```c
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],
205          lon89a[0][0]);
```

* geo1_89a[0][0] is read for layer 1.

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

```c
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   }
```

- 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.
- Scale handling is automatically applied when using AMTK. For details to P.22.
Read L2 pixel data quality * Numbers written on the left are row number of the sample program.

```c
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.

```c
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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

# source filename
csrc="readL2H_hdf5.c amsr2time_.c "

# output filename
out=readL2H_hdf5_c

# library order
lib="-lhdf5_hl -lhdf5 -lsz -lz -lm"

# c compile
cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib"
echo $cmd
$cmd

# garbage
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
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-I/home/user1/util/szip_2.1/include
-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 >
$ 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 “readL2H_hdf5_c” is shown as follows.

```bash
$ ./readL2H_hdf5_c GW1AM2_201303011809_125D_L2SGPRCHAA00000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHAA00000000.h5
GeophysicalName: Precipitation
GranuleID: GW1AM2_201303011809_125D_L2SGPRCHAA00000000
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
c.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)
geol_89a[scan=0][pixel=0]: -32767.0 [mm/h] (PDQ: 16)
geol_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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td>* Brightness Temperature (H)</td>
</tr>
<tr>
<td>- GranuleID</td>
<td>- Brightness Temperature (V)</td>
</tr>
</tbody>
</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 <string.h>
4 #include "hdf5.h"
5
6 int main(int argc, char *argv[]){
7   // interface variable
8   int i,j; // loop variable
9   herr_t ret; // return status
10  char *buf = NULL; // text buffer
11  char *fn = NULL; // filename
12  hid_t fhnd; // file handle
13  hid_t shnd; // dataspace handle
14  hid_t atyp; // attribute type
15  hid_t dhnd; // dataset handle
16  hid_t ahnd; // attribute handle
17  hsize_t szl[3]; // array size 1
18  hsize_t sz2[3]; // array size 2
19  int x; // grid size x
20  int y; // grid size y
21  void *buf = NULL;
22  void *fn = NULL;
23  void *fhnd = NULL;
24  void *shnd = NULL;
25  void *atyp = NULL;
26  void *dhnd = NULL;
27  void *ahnd = NULL;
28  void *szl = NULL;
29  void *sz2 = NULL;
30  void *x = NULL;
31  void *y = NULL;
32  // meta data
33  char *geo = NULL; // GeophysicalName
34  char *gid = NULL; // GranuleID
35
36  // array data
37  float *tbH; // brightness temperature for horizontal
38  float *tbV; // brightness temperature for vertical
39
40  // Initialize the HDF5 library.
41  Initialize the HDF5 library.
42  ret = H5open();
43  ret: [Return value] Error: A negative value is returned.
44
45  // hdf5 initialized
46  ret = H5open();
47  if(ret < 0){
48    printf("h5open error: %d\n", ret);
49    exit(1);
50  }
51
52  // Open an existing HDF5 file.
53  fhnd = H5Fopen(fn, H5F_ACC_RDONLY, H5P_DEFAULT);
54  fn: AMSR2 HDF file name.
55  label1: File access flags. H5F_ACC_RDONLY allow read-only access to file.
56  label2: Use H5P_DEFAULT for default file access properties.
57  fhnd: [Return value] Normal: HDF access file id is returned. Error: A negative value is returned.
58
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  }

Open HDF file * Numbers written on the left are row number of the sample program.
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.
\[
\text{ahnd = H5Aopen(fhnd, nam, label);} \\
\text{fhnd: Object id.} \\
\text{nam: Name of attribute.} \\
\text{label: Use H5P_DEFAULT.} \\
\text{ahnd: [Return value] Normal: Attribute id is returned. Error: A negative value is returned.}
\]

Get an attribute data type.
\[
\text{atyp = H5Aget_type(ahnd);} \\
\text{ahnd: Attribute id.} \\
\text{atyp: [Return value] Normal: Datatype id is returned. Error: A negative value is returned.}
\]

Read an attribute.
\[
\text{ret = H5Aread(ahnd, atyp, buf);} \\
\text{ahnd: Attribute id.} \\
\text{atyp: Datatype id. (Attribute is automatically converted to specified data type.)} \\
\text{buf: Buffer for data to be read.} \\
\text{ret: [Return value] Error: A negative value is returned.}
\]

```c
// read meta: GeophysicalName 
ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
atyp = H5Aget_type(ahnd);
ret = H5Aread(ahnd, atyp, &geo);
if (ret < 0)
  printf("H5Aread error: GeophysicalName\n");
ext(1);
}
ret = H5Aclose(ahnd);
printf("GeophysicalName: %s\n", geo);
```

Close the attribute.
\[
\text{ret = H5Aclose(ahnd);} \\
\text{ahnd: Attribute id.} \\
\text{ret: [Return value] Error: A negative value is returned.}
\]
Acquisition of the array size and memory allocation

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

```c
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);

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   }
```

<table>
<thead>
<tr>
<th>Get the size of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open an existing dataset.</td>
</tr>
<tr>
<td>dhnd = H5Dopen(fhnd, &quot;Brightness Temperature (H)&quot;, H5P_DEFAULT);</td>
</tr>
<tr>
<td>fhnd: HDF access file id.</td>
</tr>
<tr>
<td>nam: The name of the dataset to access.</td>
</tr>
<tr>
<td>label: Use H5P_DEFAULT.</td>
</tr>
<tr>
<td>dhnd: [Return value] Normal: Dataset id is returned. Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return an identifier for the dataspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>shnd = H5Dget_space(dhnd)</td>
</tr>
<tr>
<td>dhnd: Dataset id</td>
</tr>
<tr>
<td>shnd: [Return value] Normal: Dataspace id is returned. Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acquisition of data array dimension.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret = H5Sget_simple_extent_dims(shnd, sz1, sz2)</td>
</tr>
<tr>
<td>shnd: Dataspace id.</td>
</tr>
<tr>
<td>sz1: [Return value] Size of each dimension.</td>
</tr>
<tr>
<td>sz2: [Return value] Maximum size of each dimension.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Release the dataspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret = H5Sclose(shnd)</td>
</tr>
<tr>
<td>shnd: Dataspace id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Close the dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret = H5Dclose(dhnd);</td>
</tr>
<tr>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocate the memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>116 // memory allocate</td>
</tr>
<tr>
<td>117 tbH = malloc(sizeof(float)<em>x</em>y);</td>
</tr>
<tr>
<td>118 if(tbH == NULL){</td>
</tr>
<tr>
<td>119 printf(&quot;memory allocate error: tbH\n&quot;);</td>
</tr>
<tr>
<td>120 exit(1);</td>
</tr>
<tr>
<td>121 }</td>
</tr>
</tbody>
</table>
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.

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.

179
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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
cc=icc

csrc=readL3B_hdf5.c

out=readL3B_hdf5_c

# library order
lib="-lhdf5 HL -lhdf5 -lsz -lz -lm"

# c compile
cmd="$cc -g $csrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib"
echo $cmd
$cmd

garbage
rm -f *.o
```

The execution example of “build_readL3B_hdf5_c.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./build_readL3B_hdf5_c.sh
icc -g readL3B_hdf5.c -o readL3B_hdf5_c
-I/home/user1/util/hdf5_1.8.4-patch1/include
-L/home/user1/util/hdf5_1.8.4-patch1/lib
-I/home/user1/util/szip_2.1/include
-L/home/user1/util/szip_2.1/lib
-lhdf5 HL -lhdf5 -lsz -lz -lm
```

Specify the library directories in row number 7-8. “include” and “lib” directories are necessary under the each library directories.

Specify the compiler you use in row number 11. Intel compiler (icc), PGI compiler (pgcc), or GNU compiler (gcc) is required.
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 >
$ 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 “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)
+------------------------------------------------------------------------+
| 4112334444444444444444444444444444444444444444444444444444443222222244444 |
| 111134311134444444444444444444444444444444444444444444444434333333333 |
| 1245244555555544444444444444444443431234343441445555444444444421111111 |
| 15454444444444444444444444454111111111111111111111111111111111111111111 |
| 14224255441444454444444453111111111111111111111111455544444551111111111 |
| 444444444444444444445555441111111111111111111111111111111111111111111111 |
| 44444444444444444444444444444444444444444444444444444444444444444444444 |
| 55555555555511112222333333333333333333333333333333333333333333333333333 |
| 555555555555555555555555555555555555555555555555555555555555555555555555 |
+------------------------------------------------------------------------+
ASCII ART OF

VERTICAL BRIGHTNESS TEMPERATURE (X/20GRID Y/40GRID)

+---------------------------------------------------------------+
|                  |                  |
| 433344444444455455444444444444444444444444444444444444444444433333244545 |
| 333345333345545544444444444444444444444444444444444444444444344344333 |
| 3355345555555555555555545555455554555455555545555555555555555555555555 |
| 4333435555555555555555555555555555555555555555555555555555555555555555 |
| 5555555555555555555555555555555555555555555555555555555555555555555555 |
| 5555555555555555555555555555555555555555555555555555555555555555555555 |
| 6555666555555555555555555555555555555555555555555555555555555555555555 |
| 3355555633333333333333333333333333333333333333333333333333333333333333 |
| 3335555433333333333333333333333333333333333333333333333333333333333333 |
| 3335555333333333333333333333333333333333333333333333333333333333333333 |
| 3355555666653333333333333333333333333333333333333333333333333333333333 |
| 4443344444444344444444444444444443333333333333333333333333333333333333 |
| 4444444333333333333333333333333333333333333333333333333333333333333333 |
+---------------------------------------------------------------+

[#]: 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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* GeophysicalName</td>
<td></td>
</tr>
<tr>
<td>- GranuleID</td>
<td>* Geophysical Data</td>
</tr>
</tbody>
</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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hdf5.h"

int main(int argc, char *argv[])
{
  // interface variable
  int i, j; // loop variable
  herr_t ret; // return status
  char *buf = NULL; // text buffer
  char *fn = NULL; // filename
  hid_t fhnd; // file handle
  hid_t ahnd; // attribute handle
  hid_t atyp; // attribute type
  hid_t dhnd; // dataset handle
  hid_t shnd; // dataspace handle
  hsize_t sz1[3]; // array size 1
  hsize_t sz2[3]; // array size 2
  int x; // grid size x
  int y; // grid size y

  // meta data
  char *geo = NULL; // GeophysicalName
  char *gid = NULL; // GranuleID

  // array data
  float *geo1; // geophysical data layer 1
  float *geo2; // geophysical data layer 2
  float *geotmp; // geophysical data temporary
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hdf5.h"

int main(int argc, char *argv[])
{
  // interface variable
  int i, j; // loop variable
  herr_t ret; // return status
  char *buf = NULL; // text buffer
  char *fn = NULL; // filename
  hid_t fhnd; // file handle
  hid_t ahnd; // attribute handle
  hid_t atyp; // attribute type
  hid_t dhnd; // dataset handle
  hid_t shnd; // dataspace handle
  hsize_t sz1[3]; // array size 1
  hsize_t sz2[3]; // array size 2
  int x; // grid size x
  int y; // grid size y

  // meta data
  char *geo = NULL; // GeophysicalName
  char *gid = NULL; // GranuleID

  // array data
  float *geo1; // geophysical data layer 1
  float *geo2; // geophysical data layer 2
  float *geotmp; // geophysical data temporary
```

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

   ```c
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include "hdf5.h"

   int main(int argc, char *argv[])
   {
     // interface variable
     int i, j; // loop variable
     herr_t ret; // return status
     char *buf = NULL; // text buffer
     char *fn = NULL; // filename
     hid_t fhnd; // file handle
     hid_t ahnd; // attribute handle
     hid_t atyp; // attribute type
     hid_t dhnd; // dataset handle
     hid_t shnd; // dataspace handle
     hsize_t sz1[3]; // array size 1
     hsize_t sz2[3]; // array size 2
     int x; // grid size x
     int y; // grid size y

     // meta data
     char *geo = NULL; // GeophysicalName
     char *gid = NULL; // GranuleID

     // array data
     float *geo1; // geophysical data layer 1
     float *geo2; // geophysical data layer 2
     float *geotmp; // geophysical data temporary
   ```
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`: Attribute id. Normal: Attribute id is returned. Error: A negative value is returned.

### Get an attribute data type.
```
atyp = H5Aget_type(ahnd);
```
- `ahnd`: Attribute id.
- `atyp`: Datatype id. 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`: Error: A negative value is returned.

### Close the attribute.
```
ret = H5Aclose(ahnd);
```
- `ahnd`: Attribute id.
- `ret`: Error: A negative value is returned.

---

```c
// read meta: GeophysicalName
ahnd = H5Aopen(fhnd, "GeophysicalName", H5P_DEFAULT);
atyp = H5Aget_type(ahnd);
ret = H5Aread(ahnd, atyp, &geo);
if(ret < 0){
    printf("H5Aread error: GeophysicalName\n");
    exit(1);
}
ret = H5Aclose(ahnd);
printf("GeophysicalName: %s\n", geo);
```
Acquisition of the array size and memory allocation

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

<table>
<thead>
<tr>
<th>Get the size of the array.</th>
<th>Return an identifier for the dataspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dhnd = H5Dopen(fhnd,&quot;Geophysical Data&quot;, H5P_DEFAULT);</td>
<td>shnd = H5Dget_space(dhnd)</td>
</tr>
<tr>
<td>nam: The name of the dataset to access.</td>
<td>dhnd: Dataset id</td>
</tr>
<tr>
<td>label: Use H5P_DEFAULT.</td>
<td>shnd: [Return value] Normal: Dataspace id is returned. Error: A negative value is returned.</td>
</tr>
<tr>
<td>shnd: [Return value] Normal: Dataspace id is returned. Error: A negative value is returned.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acquisition of data array dimension.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ret = H5Sget_simple_extent_dims(shnd,sz1,sz2)</td>
<td>shnd: Dataspace id.</td>
</tr>
<tr>
<td>sz1: [Return value] Size of each dimension.</td>
<td>sz2: [Return value] Maximum size of each dimension.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
<td></td>
</tr>
</tbody>
</table>

```
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);
```

<table>
<thead>
<tr>
<th>Release the dataspace.</th>
<th>Close the dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret = H5Sclose(shnd)</td>
<td>ret = H5Dclose(dhnd);</td>
</tr>
<tr>
<td>shnd: Dataspace id.</td>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
<td></td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocate the memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>120   // memory allocate layer 1</td>
</tr>
<tr>
<td>121   geo1=malloc(sizeof(float)<em>x</em>y);</td>
</tr>
</tbody>
</table>
| 122   if(geo1==NULL){
123       printf("memory allocate error: geo1\n");
124       exit(1);
125   } |

186
Read geophysical quantities * Numbers written on the left are row number of 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”.

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.

```c
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.
```

Read scale factor.

```c
// read layer 1
if(strncmp(gid+29,"SND",3)!=0 && strncmp(gid+29,"SST",3)!=0){
dhnd = H5Dopen(fhnd, "Geophysical Data", H5P_DEFAULT);
// get scale
ahnd = H5Aopen(dhnd, "SCALE FACTOR", H5P_DEFAULT);
ret = H5Aread(ahnd, H5T_NATIVE_FLOAT, &sca);
}
```

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(strcmp(gid+29,"SND",3)==0 || strcmp(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,
172       geotmp);
173     if(ret < 0){
174       printf("H5Dread error: Geophysical Data\n");
175       exit(1);
176     }
177     ret = H5Dclose(dhnd);
178   // separate
179     for(j = 0; j < y; ++j){
180       for(i = 0; i < x; ++i){
181         geo1[x*j+i] = geotmp[x*j*2+i*2+0];
182         geo2[x*j+i] = geotmp[x*j*2+i*2+1];
183       }
184     }
185   // change scale
186     for(j = 0; j < y; ++j){
187       for(i = 0; i < x; ++i){
188         if(geo1[x*j+i] > -32767) geo1[x*j+i] = geo1[x*j+i] * sca;
189         if(geo2[x*j+i] > -32767) geo2[x*j+i] = geo2[x*j+i] * sca;
190       }
191   }

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

373   // memory free
374     free(geo1);
375   if(strcmp(gid+29,"SND",3)==0 || strcmp(gid+29,"SST",3)==0){
376     free(geo2);
377   }

Release the memory.

Close the HDF5 file.

379   // close
380     ret = H5Fclose(fhnd);
381     ret = H5close();

Flush all data to disk, close all open identifiers, and clean up memory.

Termiate access to an HDF5 file.

ret = H5Fclose(fhnd);
fhnd: HDF access file id.
ret: [Return value] Error: A negative value is returned.
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
-LSZIP/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 >
$ 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 “readL3G_hdf5_c” is shown as follows.

```bash
$ ./readL3G_hdf5_c GW1AM2_20130200_01M_EQMA_L3SGCLWLAL0000000.h5
```

input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLAL0000000.h5
GeophysicalName: Cloud Liquid Water
GranuleID: GW1AM2_20130200_01M_EQMA_L3SGCLWLAL0000000
grid size x: 1440
grid size y: 720

ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/20GRID Y/40GRID)

[...]
```
[#]: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)

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>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>* NumberOfScans are acquired from the array dimension of the dataset.</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>Fixed values are used for the variables below.</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* OverlapScans</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td>* CoRegistrationParameterA1</td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* CoRegistrationParameterA2</td>
<td>* Brightness Temperature (6.9GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (6.9GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (7.3GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (7.3GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (10.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (10.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (18.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (18.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (23.8GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (23.8GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (36.5GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (36.5GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-A,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-A,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-B,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (89.0GHz-B,V)</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality 6 to 36</td>
</tr>
<tr>
<td></td>
<td>- Pixel Data Quality 89</td>
</tr>
<tr>
<td></td>
<td>* Land_Ocean Flag 6 to 36</td>
</tr>
<tr>
<td></td>
<td>- Land_Ocean Flag 89</td>
</tr>
<tr>
<td></td>
<td>* Earth Incidence</td>
</tr>
<tr>
<td></td>
<td>- Earth Azimuth</td>
</tr>
</tbody>
</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.
AMSR2 data users manual

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

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 defined 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 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.

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

**Initialize the HDF5 library.**

```fortran
  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.**

```fortran
  114 C open
  115    call H5open_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
  116    if(ret.lt.0)then
  117      write(*,'(a,a)')'H5open error: ',fn(1:len_trim(fn))
  118      call exit(1)
  119    endif
```

Initialize the HDF5 library.

call H5open_f(ret)

ret: [Return value] Error: A negative value is returned.

Open an existing HDF5 file.

call H5open_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.
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 datasource 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 datasource.
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.

Release the datasource.
call H5Dclose_f(dhnd,ret)  
dhnd: Dataset 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.

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.

```
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: ','Scan Time'
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,sta)
185     ! sample display
186     write(*,'(a,i4.4,"/",i2.2,"/",i2.2,"/",i2.2,"/",i2.2,":",i2.2,":",i2.2)')
187     +"time(scan=1): "
188     +,sta(1)%year
189     +,sta(1)%month
190     +,sta(1)%day
191     +,sta(1)%hour
192     +,sta(1)%minute
193     +,sta(1)%second
```

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.

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.

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.
AMS R2 has frequency channels of 6, 7, 10, 18, 23, 36, and 89 GHz and their observation point are not strictly same. Latitude and longitude stored in L1B products are that of 89 GHz. Position of low frequency data can be calculated from position of 89 GHz-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.

AMS R2 data users manual

Read latitude and longitude

Read latitude and longitude of 89 GHz. Overlap scans contained at both end of L1 data should be removed. Latitudes and longitudes of low frequency data are calculated from position of 89 GHz-A and the co-registration parameters.

Read latitude.

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)
233 : Read longitude.
234 C read array: latlon for low mean
235 call amsr2latlon(num,prm1(7),prm2(7),lat89a,lon89a,latlm,lonlm)
236 write(*,'(a,"(",f9.4,"",",f9.4,"")")')'latlonlm(pixel=1,scan=1): '
237 +,latlm(1,1),lonlm(1,1)

AMS R2 has frequency channels of 6, 7, 10, 18, 23, 36, and 89 GHz and their observation point are not strictly same. Latitude and longitude stored in L1B products are that of 89 GHz. Position of low frequency data can be calculated from position of 89 GHz-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.

Remove overlap scans.

Calculate 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 89 GHz-A.
lon89a: Longitude of 89 GHz-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.
atyp: 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) ! get scale
305 call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret) ! get scale
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,num
320 if(tb06h(i,j).lt.65534)tb06h(i,j)=tb06h(i,j)*sca
321 endif
322 enddo
323 Remove overlap scans.
324 do j=1,num
325 ! sample display
326 enddo
327
328 "tb06h(pixel=1,scan=1): ",tb06h(1,1)

Scale handling is automatically applied when using AMTK. For details to P.22. 

Close the attribute.
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.
Read L1B pixel data quality

“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.

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.
AMSR2 data users manual

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)

“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
```

Read L1B land ocean flag.

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.

```fortran
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) ! get scale
940     call H5Aclose_f(ahnd,ret) ! get scale
941 szl(1)=LMT
942 szl(2)=AM2_DEF_SNUM_LO
943 call H5Dread_f(dhnd + ,H5T_NATIVE_REAL,ear_in,szl,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)

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

Close the HDF5 file.

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

Specify the library directories in row number 7-8. “include” and “lib” directories are necessary under the each library directories.

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.

The execution example of “build_readL1B_hdf5_f.sh” is shown in the following.

```
$ ./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
-l/home/user1/util/hdf5_1.8.4-patch1/include
-l/home/user1/util/hdf5_1.8.4-patch1/lib
-l/home/user1/util/szip_2.1/include
-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 >
$ 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_hdf5_f” is shown as follows.

```
$ ./readL1B_hdf5_f GW1AM2_201207261145_055A_L1S
GBTBR_000000.h5
input file: GW1AM2_201207261145_055A_L1SGTBTR_000000.h5
NumberOfScans(RETRIEVE BY ARRAY SIZE): 19
79
OverlapsScans(FIXED VALUE): 20
limit of NumberOfScans = 2200
CoRegistrationParameter1(FIXED VALUE): 6G-1.25 000,7G-1.25000,18G-1.25000,23G-1.25000,36G-1.00000
CoRegistrationParameter2(FIXED VALUE): 6G-0.00 000,7G--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)
latlon10(pixel=1,scan=1): (-73.3592, 136.6213)
latlon18(pixel=1,scan=1): (-73.3506, 136.6001)
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
... (other variables listed below)...
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): 186.39
... (other variables listed below)...
tb89ah(pixel=1,scan=1): 168.16
```

The example continues with the rest of the variables listed in the output.
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>* NumberOfScans are acquired from the array dimension of the dataset.</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>Fixed values are used for the variable below.</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* OverlapScans</td>
<td>- Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td></td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td></td>
<td>* Brightness Temperature (res06,6.9GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res06,6.9GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res06,7.3GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res06,7.3GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res10,10.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,18.7GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res23,23.8GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,36.5GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (res36,89.0GHz,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-A,V)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,H)</td>
</tr>
<tr>
<td></td>
<td>- Brightness Temperature (original,89GHz-B,V)</td>
</tr>
<tr>
<td>Data extracted from observation point of 89GHz-A</td>
<td>* Pixel Data Quality 6 to 36</td>
</tr>
<tr>
<td>* Latitude and longitude of low frequency data</td>
<td>- Pixel Data Quality 89</td>
</tr>
<tr>
<td>* Pixel Data Quality 6 to 36</td>
<td>* Land_Ocean Flag 6 to 36</td>
</tr>
<tr>
<td>* Land_Ocean Flag 89</td>
<td>- Land_Ocean Flag 89</td>
</tr>
<tr>
<td>* Earth Incidence</td>
<td>* Earth Azimuth</td>
</tr>
</tbody>
</table>

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
AMS R2 data users manual

**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
```

**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 defined 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.

**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 the variables for metadata.**

※ Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library. Fixed values are set in the program.

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

Initialize the HDF5 library.

```fortran
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 an existing HDF5 file.

```fortran
102 C open
103       call H5Open_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,label2)
104       if(ret.lt.0)then
105         write(*,'(a,a)')'H5open error: ',fn(1:len_trim(fn))
106        call exit(1)
107       endif
```

- **Initialize the HDF5 library.**
  - **call H5open_f(ret)**
  - ret: [Return value] Error: A negative value is returned.

- **Open the HDF5 file.**
  - **call H5Open_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.
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.
```fortran
    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.
```fortran
    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.
```fortran
    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.
```

Release the dataspace.
```fortran
    call H5Sclose_f(shnd,ret)
    shnd: Dataspace id.
    ret: [Return value] Error: A negative value is returned.
```

Close the dataset.
```fortran
    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.

**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.

```c
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,:",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
```

**For details to P.13.**

Read scanning time.

For details to P.13.

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.

```fortran
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
```

For details to P.20.

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).

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.

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.
atyp: Datatype id. (Attribute is automatically converted to specified data type.)
buf: Buffer for data to be read.
sz1: [Return value] Error: A negative value is returned.

Read scale factor.
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)

Read brightness temperature.
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd,+,'Brightness Temperature (res06,6.9GHz,H)',tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
  write(*,'(a,a)')'H5Dread error: '
  write(*,1)'
182 ','Brightness Temperature (res06,6.9GHz,H)'
  call exit(1)
endif
call H5Dclose_f(dhnd,ret)
!
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd,+,'Brightness Temperature (res06,6.9GHz,H)',tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
  write(*,'(a,a)')'H5Dread error: '
  write(*,1)'
182 ','Brightness Temperature (res06,6.9GHz,H)'
  call exit(1)
endif
call H5Dclose_f(dhnd,ret)
!
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd,+,'Brightness Temperature (res06,6.9GHz,H)',tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
  write(*,'(a,a)')'H5Dread error: '
  write(*,1)'
182 ','Brightness Temperature (res06,6.9GHz,H)'
  call exit(1)
endif
call H5Dclose_f(dhnd,ret)
!
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd,+,'Brightness Temperature (res06,6.9GHz,H)',tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
  write(*,'(a,a)')'H5Dread error: '
  write(*,1)'
182 ','Brightness Temperature (res06,6.9GHz,H)'
  call exit(1)
endif
call H5Dclose_f(dhnd,ret)
!
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd,+,'Brightness Temperature (res06,6.9GHz,H)',tb06h06,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
  write(*,'(a,a)')'H5Dread error: '
  write(*,1)'
182 ','Brightness Temperature (res06,6.9GHz,H)'
  call exit(1)
endif
call H5Dclose_f(dhnd,ret)
!
call H5Dopen_f(fhnd,+,'Brightness Temperature (res06,6.9GHz,H)',dhnd,ret)
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)
Read L1R pixel data quality * Numbers written on the left are row number of the sample program.

```
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)

```

**“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.**

```
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((i-1)*2+1,j+ovr),6))pdq07h(i,j)=1
825         if(btest(i4d2hi((i-1)*2+1,j+ovr),7))pdq07h(i,j)=10
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     pdq06v(:,j)=0
832     pdq06v(:,j)=0
833     pdq07v(:,j)=0
834     pdq07v(:,j)=0
835     enddo
```

**“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.
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). 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. 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       szl(1)=LMT*6
880       szl(2)=AM2_DEF_SNUM_LO
881       call H5Dread_f(dhnd,
882         +,H5T_NATIVE_INTEGER,loflo,szl,ret,H5S_ALL_F,H5S_ALL_F)
883       if(ret.lt.0)then
884         write(*,'(a,a)')'H5Dread error: '
885         write(*,'(a,a)')'Land_Ocean Flag 6 to 36'
886         call exit(1)
887       endif
888       call H5Dclose_f(dhnd,ret)

“Land_Ocean Flag 6 to 36” and “Land_Ocean Flag 89” is acquired as one data, respectively.
Split the data to each frequency.

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
```

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.

```fortran
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) ! get scale
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         call exit(1)
967     endif
968     call H5Dclose_f(dhnd,ret)
969     ! cutoff overlap & change scale
970     do j=1,num
971         do i=1,AM2_DEF_SNUM_LO
972             ear_in(i,j)=ear_in(i,j+ovr)
973             if(ear_in(i,j).gt.-32767)ear_in(i,j)=ear_in(i,j)*sca
974         enddo
975     enddo
976     do j=num+1,LMT
977         ear_in(:,j)=0
978     enddo
979     ! sample display
980     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.

```fortran
1012   call H5Fclose_f(fhnd,ret)
1013   call H5close_f(ret)
1014   end
```

Terminates access to an HDF5 file.

```
ret = H5Fclose(hfnd);
```

Flush all data to disk, close all open identifiers, and clean up memory.

```
ret = H5close();
```

flush: HDF access file id.

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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort
c=icc

# source filename
fsrc="readL1R_hdf5.f"
csrc="amsr2time_.c"
obj=`echo $csrc|sed "s/¥.c/.o/g"`

# output filename
out=readL1R_hdf5_f

# library order
lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"

# c compile
cmd="$cc -g -c $csrc"
echo $cmd
$cmd

# f compile
fsrc=$fsrc
fsrcobj=$obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib
fsrcobj $lib

echo $cmd
$cmd

# garbage
rm -f *
```

The execution example of “build_readL1R_hdf5_f.sh” is shown in the following.  
* Line feeds are inserted for convenience.

```bash
$ ./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
-I/home/user1/util/hdf5_1.8.4-patch1/lib
-I/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 >
$ 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):         19
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=   156384
 01.00
amsr2time: year=1994 month= 7 tai93sec=    471744
 02.00
amsr2time: year=1996 month= 7 tai93sec=   1418688
 04.00
amsr2time: year=1999 month= 1 tai93sec=   946080
 03.00
amsr2time: year=2006 month= 7 tai93sec=  1893024
 05.00
amsr2time: year=2009 month= 1 tai93sec=  5049216
 07.00
amsr2time: year=2012 month= 7 tai93sec=  6152544
 08.00
amsr2time: number of leap second = 8
time(scan=1): 2012/07/26 11:45:43
latlon89ar(pixel=1,scan=1): ( -73.3581, 136.8432)
tl89br(pixel=1,scan=1): ( -73.4128, 137.2216)
latlon89br(pixel=1,scan=1): ( -73.3581, 136.8432)
latlon89ar(pixel=1,scan=1): ( -73.3581, 136.8432)
tb06h06(pixel=1,scan=1):  170.40
tb07v06(pixel=1,scan=1):  198.47
tb10h10(pixel=1,scan=1):  207.11
tb10v10(pixel=1,scan=1):  207.11
tb18h23(pixel=1,scan=1):  195.90
```
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td></td>
</tr>
<tr>
<td>* NumberOfScans are acquired from the array dimension of the dataset.</td>
<td></td>
</tr>
<tr>
<td>Fixed values are used for the variable below. * OverlapScans</td>
<td></td>
</tr>
<tr>
<td>* Scan Time</td>
<td></td>
</tr>
<tr>
<td>* Latitude of Observation Point</td>
<td></td>
</tr>
<tr>
<td>* Longitude of Observation Point</td>
<td></td>
</tr>
<tr>
<td>* Geophysical Data</td>
<td></td>
</tr>
<tr>
<td>* Pixel Data Quality</td>
<td></td>
</tr>
</tbody>
</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'

C fixed value
6. integer(4),parameter::LMT=2200 ! limit of NumberOfScans
7. integer(4),parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
8. integer(4),parameter::AM2_DEF_SNUM_LO=243 ! low resolution pixel width

C interface variable
9. integer(4) i,j          ! loop variable
10. integer(4) ret          ! return status
11. character(len=512) buf  ! text buffer
12. character(len=512) fn   ! filename
13. integer(HID_T) fhnd     ! file handle
14. integer(HID_T) ahnd     ! attribute handle
15. integer(HID_T) atyp     ! attribute type
16. integer(HID_T) dhnd     ! dataset handle
17. integer(HID_T) shnd     ! dataspace handle
18. integer(HSIZE_T) sz1(3) ! array size 1
19. integer(HSIZE_T) sz2(3) ! array size 2
20. integer(4) num          ! NumberOfScans
21. integer(4) ovr          ! OverlapScans
22. parameter(ovr=0)

C array data
23. type(AM2_COMMON_SCANTIME) st(LMT) ! scantime
24. real(4) lat(AM2_DEF_SNUM_LO,LMT) ! lat
25. real(4) lon(AM2_DEF_SNUM_LO,LMT) ! lon
26. real(4) geo1(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 1
27. real(4) geo2(AM2_DEF_SNUM_LO,LMT) ! geophysical data layer 2
28. real(4) geotmp(AM2_DEF_SNUM_LO,LMT*2) ! geophysical data temporary
29. integer(4) pdq1(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 1
30. integer(4) pdq2(AM2_DEF_SNUM_LO,LMT) ! pixel data quality layer 2
31. integer(4) pdqtmp(AM2_DEF_SNUM_LO,LMT*2) ! pixel data quality temporary

AMSR2 data users manual

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.

Define the variables for datasets.

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

```c
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.

```c
67 C open
68   call H5Fopen_f(fn,label1,fhnd,ret,label2)
69   if(ret.lt.0)then
70     write(*,'(a,a)')'H5Fopen error: ',fn(1:len_trim(fn))
71     call exit(1)
72   endif
```

---

**Initialize the HDF5 library.**

```c
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 an existing HDF5 file.**

```c
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
```

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.
```

**NumberOfScans** is represented as below.

Sum of scans in product - (OverlapScans x 2)

```
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(fhnd,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.

\[
\begin{align*}
1 & \text{C read array: scantime} \\
2 & \text{read} \\
3 & \quad \text{call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)} \\
4 & \quad \text{sz1(1)=LMT} \\
5 & \quad \text{call H5Dread_f(dhnd,} \\
6 & \quad \text{+,'H5T_NATIVE_DOUBLE',r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F} \\
7 & \quad \text{if(ret.lt.0)then} \\
8 & \quad \text{write('(','(a,a)',')'H5Dread error: ','Scan Time'} \\
9 & \quad \text{call exit(1)} \\
10 & \quad \text{endif} \\
11 & \quad \text{call H5Dclose_f(dhnd,ret)} \\
12 & \text{! convert} \\
13 & \text{call amsr2time(num,r8d1,st)} \\
14 & \text{! sample display} \\
15 & \text{write(*,'('a,i4.4,"/","i2.2,"/","i2.2","_i2.2","_i2.2":"_i2.2')')} \\
16 & \quad +\text{time(scan=1): '}
17 & \quad +,st(1)%year \\
18 & \quad +,st(1)%month \\
19 & \quad +,st(1)%day \\
20 & \quad +,st(1)%hour \\
21 & \quad +,st(1)%minute \\
22 & \quad +,st(1)%second
\end{align*}
\]

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

```fortran
133 C read array: latlon
134       ! read lat
call H5Dopen_f(fhnd,'Latitude of Observation Point',dhnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd
+,'H5T_NATIVE_REAL,lat,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
   write(*,'(a,a)')'H5Dread error: '
   +,'Latitude of Observation Point'
call exit(1)
endif
136       ! read lon
137       call H5Dopen_f(fhnd,'Longitude of Observation Point',dhnd,ret)
sz1(1)=LMT
sz1(2)=AM2_DEF_SNUM_LO
call H5Dread_f(dhnd
+,'H5T_NATIVE_REAL,lon,sz1,ret,H5S_ALL_F,H5S_ALL_F)
if(ret.lt.0)then
   write(*,'(a,a)')'H5Dread error: '
   +,'Longitude of Observation Point'
call exit(1)
endif
139       ! sample display
140       write(*,'(a,"(,f9.4,,f9.4,"\")")')'latlon(pixel=1,scan=1): '
141       +,lat(1,1),lon(1,1)
```

Read latitude and longitude. For details to P.20
Read geophysical quantities * Numbers written on the left are row number of the sample program.

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.  
atyp: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.

---

Exclude the missing value (-32768) and error value (-32767) when scale handling is applied.

---

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”.

---

Read geophysical quantities.

Read scale factor.

Read geophysical quantity.

Scale handling is automatically applied when using AMTK. For details to P.22.
When product has 2 layers, read all data at once in temporary variable and then separate for each layer.

```fortran
C read array: geophysical data for 2 layer
186 if((gid(30:32).eq.'SND').or.(gid(30:32).eq.'SST'))then
187     call H5Dopen_f(fhnd,'Geophysical Data',dhnd,ret)
188     ! get scale
189     call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
190     call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
191     call H5Aclose_f(ahnd,ret)
192     ! read
193     sz1(1)=LMT*2
194     sz1(2)=AM2_DEF_SNUM_LO
195     call H5Dread_f(dhnd
196     + ,H5T_NATIVE_REAL,geotmp,sz1,ret,H5S_ALL_F,H5S_ALL_F)
197     if(ret.lt.0)then
198         write(*,'(a,a)')'H5Dread error: '
199         call exit(1)
200     endif
201     call H5Dclose_f(dhnd,ret)
202     ! separate & change scale
203     do j=1,num
204         geo1(i,j)=geotmp((i-1)*2+1,(j-1)*2+1)
205         geo2(i,j)=geotmp((i-1)*2+2,(j-1)*2+2)
206         if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
207         if(geo2(i,j).gt.-32767)geo2(i,j)=geo2(i,j)*sca
208     enddo
209 endif
210```

186 C read array: geophysical data for 2 layer
**AMSR2 data users manual**

**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     call exit(1)
224   endif
225   call H5Dclose_f(dhnd,ret)
226 endif

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     call exit(1)
239   endif
240   ! separate
241   call H5Dclose_f(dhnd,ret)
242   do j=1,num
243     !
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

---

224
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.

```bash
#!/bin/sh

### environment
export LANG=C

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort
cc=icc

# source filename
fsrc="readL2L_hdf5.f"
csrc="amsr2time_.c"
obj=`echo $csrc|sed "s/.*c/.o/g"`

# output filename
out=readL2L_hdf5_f

# library order
lib="-lhdf5hl_fortran -lhdf5_hl -lhdf5_fortran -lhdf5 -lsz -lz -lm"

# c compile
cmd="$cc -g -c $csrc"
echo $cmd
$cmd

# f compile
cmd="$fc -g $fsrc $obj -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib"
echo $cmd
$cmd

# garbage
rm -f *.o
```

The execution example of “build_readL2L_hdf5_f.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./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/hdf5_1.8.4-patch1/lib
-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.

\[
\begin{align*}
< & \text{For csh or tcsh }> \\
\$ \text{unlimit} & \\
< & \text{For sh or bash }> \\
\ast & \text{Type the following four commands in order.} \\
\$ \text{ulimit -d unlimited} & \\
\$ \text{ulimit -m unlimited} & \\
\$ \text{ulimit -s unlimited} & \\
\$ \text{ulimit -v unlimited} & \\
\end{align*}
\]

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_L2SGCLWLA0000000
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=410272206.00
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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td>* Scan Time</td>
</tr>
<tr>
<td>* NumberOfScans are acquired from the array dimension of the dataset.</td>
<td>* Latitude of Observation Point for 89A</td>
</tr>
<tr>
<td>Fixed values are used for the variable below.</td>
<td>- Latitude of Observation Point for 89B</td>
</tr>
<tr>
<td>* OverlapScans</td>
<td>* Longitude of Observation Point for 89A</td>
</tr>
<tr>
<td></td>
<td>- Longitude of Observation Point for 89B</td>
</tr>
<tr>
<td></td>
<td>* Geophysical Data for 89A</td>
</tr>
<tr>
<td></td>
<td>- Geophysical Data for 89B</td>
</tr>
<tr>
<td></td>
<td>* Pixel Data Quality for 89A</td>
</tr>
<tr>
<td></td>
<td>- Pixel Data Quality for 89B</td>
</tr>
</tbody>
</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       include 'amsr2time_f.h'
5       integer(4), parameter::LMT=2200 ! limit of NumberOfScans
6       integer(4), parameter::AM2_DEF_SNUM_HI=486 ! high resolution pixel width
7       integer(4), parameter::AM2_DEF_SNUM_LO=243 ! low resolution pixel width
8       integer(4) i,j          ! loop variable
9       integer(4) ret          ! return status
10      character(len=512) buf  ! text buffer
11      character(len=512) fn   ! filename
12      integer(HID_T) fhnd     ! file handle
13      integer(HID_T) ahnd     ! attribute handle
14      integer(HID_T) atyp     ! attribute type
15      integer(HID_T) dhnd     ! dataset handle
16      integer(HID_T) shnd     ! dataspace handle
17      integer(HSIZE_T) sz1(3) ! array size 1
18      integer(HSIZE_T) sz2(3) ! array size 2
19      integer(4) num          ! NumberOfScans
20      integer(4) ovr          ! OverlapScans
21      parameter(ovr=0)

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 defined 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.

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.

Define the variables for datasets.

Employ the USE statement to make use of the HDF5 functions in the program.

Include header files of subroutines for time conversion.
Open HDF file * Numbers written on the left are row number of the sample program.

```fortran
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 an existing HDF5 file.
```

```fortran
65 C open
66         call H5Open_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
67         if(ret.lt.0)then
68           write(*,'(a,a)')'H5open error: ',fn(1:len_trim(fn))
69           call exit(1)
70         endif
```

Initialize the HDF5 library.

```
Initialize the HDF5 library.
call H5open_f(ret)
ret: [Return value] Error: A negative value is returned.
```

Open the HDF5 file.

```
Open an existing HDF5 file.
call H5open_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.
```

Initialize the HDF5 library.
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
```

**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(fhnd,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
105       call H5Dopen_f(fhnd,'Scan Time',dhnd,ret)
106       sz1(1)=LMT
107       call H5Dread_f(dhnd,H5T_NATIVE_DOUBLE,r8d1,sz1,ret,H5S_ALL_F,H5S_ALL_F)
108       if(ret.lt.0)then
109         write(*,'(a,a)')'H5Dread error: ','Scan Time'
110           call exit(1)
111       endif
112       call H5Dclose_f(dhnd,ret)
113       ! convert
114       call amsr2time(num,r8d1,st)
115       ! sample display
116       write(*,'(a,i4.4,"/",i2.2,"/",i2.2," ",i2.2," ",i2.2," ",i2.2)')
117       +'time(scan=1): '
118       +,st(1)%year
119       +,st(1)%month
120       +,st(1)%day
121       +,st(1)%hour
122       +,st(1)%minute
123       +,st(1)%second

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)
um: 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.

125 C read array: latlon for 89a
126 ! read lat
127 call HDFopen_f(fhnd
128 +,'Latitude of Observation Point for 89A',dhnd,ret)
129 sz1(1)=LMT
130 sz1(2)=AM2_DEF_SNUM_HI
131 call HDFread_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 HDFclose_f(dhnd,ret)
139 ! read lon
140 call HDFopen_f(fhnd
141 +,'Longitude of Observation Point for 89A',dhnd,ret)
142 sz1(1)=LMT
143 sz1(2)=AM2_DEF_SNUM_HI
144 call HDFread_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 HDFclose_f(dhnd,ret)
152 ! sample display
153 write(*,'("(f9.4,"",f9.4,"")")')'latlon89a(pixel=1,scan=1): '
154 +,lat89a(1,1),lon89a(1,1)
Read geophysical quantities * Numbers written on the left are row number of the sample program.

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.

```c
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
192       sz1(1)=LMT
193       sz1(2)=AM2_DEF_SNUM_HI
194       call H5Dread_f(dhnd,H5T_NATIVE_REAL,geo1_89a,sz1,ret,H5S_ALL_F,H5S_ALL_F)
195       if(ret.lt.0)then
196         write(*,'(a,a)')'H5Dread error: '
197         'Geophysical Data for 89A'
198         call exit(1)
199       endif
200       ! change scale
201       do j=1,num
202         do i=1,AM2_DEF_SNUM_HI
203           if(geo1_89a(i,j).gt.-32767)geo1_89a(i,j)=geo1_89a(i,j)*sca
204         enddo
205       enddo
```

**Read geophysical quantity.**

**Open an attribute.**
```c
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.**
```c
call H5Aclose_f(ahnd,ret)
ahnd: Attribute id.
ret: [Return value] Error: A negative value is returned.
```

**Read an attribute.**
```c
call H5Aread_f(ahnd,atyp,buf,sz1,ret)
ahnd: Attribute id.
atyp: 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.
```

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.

```
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.

```
261 C close
262     call H5Fclose_f(fhnd,ret)
263     call H5close_f(ret)
```

Flush all data to disk, close all open identifiers, and clean up memory.

```
 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
6 # library directory
7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
8 SZIP=/home/user1/util/szip_2.1
9
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
-I/home/user1/util/hdf5_1.8.4-patch1/lib
-I/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 >
$ 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 “readL2H_hdf5_f” is shown as follows.

```
$ ./readL2H_hdf5_f GW1AM2_201303011809_125D_L2SGPRCHAO00000000.h5
input file: GW1AM2_201303011809_125D_L2SGPRCHAO00000000.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_201303011809_125D_L2SGPRCHAO00000000
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
c.dat
amsr2time: year=1993 month=7 tai93sec=15638401.00
amsr2time: year=1994 month=7 tai93sec=47174402.00
amsr2time: year=1996 month=1 tai93sec=94608005.00
amsr2time: year=1997 month=7 tai93sec=141868804.00
amsr2time: year=1999 month=1 tai93sec=189302405.00
amsr2time: year=2006 month=1 tai93sec=41027206.00
amsr2time: year=2009 month=1 tai93sec=504921607.00
amsr2time: year=2012 month=7 tai93sec=615254408.00
amsr2time: number of leap second = 8
latlon89a(pixel=1,scan=1): ( 84.4188, -77.9502)
latlon89b(pixel=1,scan=1): ( 84.3305, -78.8925)
geol_89a(pixel=1,scan=1): -32767.0 [mm/h] (PDQ: 16)
geol_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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td>* Brightness Temperature (H) - Brightness Temperature (V)</td>
</tr>
</tbody>
</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.

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
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  C     character(len=512) gid  ! GranuleID

35  C array data
36  real(4),allocatable::tbH(:,:)
37  real(4),allocatable::tbV(:,:)
```

**Open HDF file**

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

```
Open HDF file.

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,a)')'(h5open_f error: ',ret
50  call exit(1)
51  endif

Open the HDF5 file.

call H5Open_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 H5open_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
54  if(ret.lt.0)then
55     write(*,'(a,a)')'(H5open 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.

<table>
<thead>
<tr>
<th>Get the size of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open an existing dataset.</strong></td>
</tr>
<tr>
<td>call H5Dopen_f(fhnd,nam,dhnd,ret)</td>
</tr>
<tr>
<td>fhnd: HDF access file id.</td>
</tr>
<tr>
<td>nam: The name of the dataset to access.</td>
</tr>
<tr>
<td>dhnd: [Return value] Dataset id.</td>
</tr>
<tr>
<td><strong>Return an identifier for the dataspace.</strong></td>
</tr>
<tr>
<td>call H5Dget_space_f(dhnd,shnd,ret)</td>
</tr>
<tr>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>shnd: [Return value] Dataspace id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acquisition of data array dimension.</th>
</tr>
</thead>
<tbody>
<tr>
<td>call H5Sget_simple_extent_dims_f(shnd,sz1,sz2,ret)</td>
</tr>
<tr>
<td>shnd: Dataspace id.</td>
</tr>
<tr>
<td>sz1: [Return value] Size of each dimension.</td>
</tr>
<tr>
<td>sz2: [Return value] Maximum size of each dimension.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Release the dataspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>call H5Sclose_f(shnd,ret)</td>
</tr>
<tr>
<td>shnd: Dataspace id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Close the dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>call H5Dclose_f(dhnd,ret)</td>
</tr>
<tr>
<td>dhnd: Dataset id.</td>
</tr>
<tr>
<td>ret: [Return value] Error: A negative value is returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allocate the memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocate(tbH(x,y),stat=ret)</td>
</tr>
<tr>
<td>if(ret.ne.0) then</td>
</tr>
<tr>
<td>write(*,'(a)') 'memory allocate error: tbH'</td>
</tr>
<tr>
<td>call exit(1)</td>
</tr>
<tr>
<td>endif</td>
</tr>
</tbody>
</table>

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<0) then
80 write(*,'(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

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.

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)
107    ! read
108    sz1(1)=y
109    sz1(2)=x
110    call H5Dread_f(dhnd,H5T_NATIVE_REAL,tbH,sz1,ret,H5S_ALL_F,H5S_ALL_F)
111    if(ret.lt.0)then
112       write(*,'(a,a)')'H5Dread error: '
113          ,'Brightness Temperature (H)'
114         call exit(1)
115    endif
116    call H5Dclose_f(dhnd,ret)
117    ! change scale
118    do j=1,y
119       do i=1,x
120          if(tbH(i,j).lt.65534)tbH(i,j)=tbH(i,j)*sca
121     enddo
122    enddo

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.
atyp: 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 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.

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.

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.

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

<table>
<thead>
<tr>
<th>Release the memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>239 C memory free</td>
</tr>
<tr>
<td>240 dealloc(tbh)</td>
</tr>
<tr>
<td>241 dealloc(tbv)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Close the HDF5 file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>242 C close</td>
</tr>
<tr>
<td>243 call H5Fclose_f(fhnd, ret)</td>
</tr>
<tr>
<td>244 call H5close_f(ret)</td>
</tr>
<tr>
<td>245 end</td>
</tr>
</tbody>
</table>

- **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.
8.5.2 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.

```bash
1 #!/bin/sh
2 3 ### environment
4 export LANG=C
5
6 # library directory
7 HDF5=/home/user1/util/hdf5_1.8.4-patch1
8 SZIP=/home/user1/util/szip_2.1
9

Specify the library directories in row number 7-8. 
“include” and “lib” directories are necessary under the each library directories.

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="-lhdf5.hl_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 *.o
```

The execution example of “build_readL3B_hdf5_f.sh” is shown in the following.

* Line feeds are inserted for convenience.

```bash
$ ./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
-lhdf5.hl_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 >
$ 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 “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)
+------------------------------------------------------------------------+
| 41123344434444444444444444444444444444444444444444444444444334432222244444 |
| 1113431111344444444444444444444444444444444444444444444444434233343111 |
| 1245234555555555554444444444444444444444444444444444444444421211111111 |
| 15454444444444444444444444444444444444444444444444444444444221111111111 |
| 41224255414444444444444453413111111111111111111111111111455544444551111111115 |
| 4444444442444455554455554411111111111111111111111111111111111111111111124 |
| 44444444444444444111551553111111111111111111111111111111111111111111111144 |
| 55555555555555551111111111111111111111111111111111111111111111111111111115 |
| 1155555551111111111111111111111111111111111111111111111111111111111111111115 |
| 1115555551111111111111111111111111111111111111111111111111111111111111111111 |
| 111555521511111111111111111111111111111111111111111111111111111111111111111 |
| 1111555111111111111111111111111111111111111111111111111111111111111111111111 |
| 1111111111111111111111111111111111111111111111111111111111111111111111111111 |
| 1111111111111111111111111111111111111111111111111111111111111111111111111111 |
| 1111111111111111111111111111111111111111111111111111111111111111111111111111 |
| 1343234443333242443434444444444444443321111111111111111111111134444444111113 |
| 33333333333333333333333333333333333333333332222223333343433333333333333333 |
+------------------------------------------------------------------------+
```
ASCII ART OF VERTICAL BRIGHTNESS TEMPERATURE (X/ 20GRID Y/ 40GRID)

+---------------------------------------------------------------+
|                                                               |
| 433344444444544444444444444444444444444444444444444444444444454 |
| 333345334554444444444444444444444444444444444444444444444445455 |
| 333554555555555554555455555455555555555555555555555555555555555 |
| 433343555555555555555555555555555555555555555555555555555555555 |
| 555555555555555555555555555555555555555555555555555555555555555 |
| 655566555555555555555555555555555555555555555555555555555555555 |
| 755565555555555555555555555555555555555555555555555555555555555 |
| 855555555555555555555555555555555555555555555555555555555555555 |
| 955555555555555555555555555555555555555555555555555555555555555 |
| 055555555555555555555555555555555555555555555555555555555555555 |
| 433344444444444444444444444444444444444444444444444444444444454 |
| 333345334554444444444444444444444444444444444444444444444445455 |
| 333554555555555554555455555455555555555555555555555555555555555 |
| 433343555555555555555555555555555555555555555555555555555555555 |
| 555555555555555555555555555555555555555555555555555555555555555 |
| 655566555555555555555555555555555555555555555555555555555555555 |
| 755565555555555555555555555555555555555555555555555555555555555 |
| 855555555555555555555555555555555555555555555555555555555555555 |
| 955555555555555555555555555555555555555555555555555555555555555 |
| 055555555555555555555555555555555555555555555555555555555555555 |
+---------------------------------------------------------------+

[#]: missing  
[O]: 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.

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Metadata stored as variable-length string cannot be acquired in HDF5 FORTRAN90 library.</td>
<td>* Geophysical Data</td>
</tr>
</tbody>
</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
**AMSR2 data users manual**

**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    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.*

```
166  C hdf5 initialize
167    call H5open_f(ret)
168    if(ret.lt.0) then
169      write(*,'(a,i12)') 'h5open_f error: ', ret
170      call exit(1)
171    endif

248  C open
249    call H5Fopen_f(fn,H5F_ACC_RDONLY_F,fhnd,ret,H5P_DEFAULT_F)
250    if(ret.lt.0) then
251      write(*,'(a,a)') 'H5Fopen error: ', fn(1: len_trim(fn))
252      call exit(1)
253    endif
```

*Initialize the HDF5 library.*

*Initialize the HDF5 library.*

*Open an existing HDF5 file.*

*Open the HDF5 file.*

[247]
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,a)')'grid size x: ',x
93 write(*,'(a,a)')'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(geol(x,y),stat=ret)
96 if(ret.ne.0) then
97 write(*,'(a)')'memory allocate error: geol'
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.
atyp: 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.

Read scale factor.
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Aclose_f(ahnd,ret)

Read geophysical quantity.
call H5Aopen_f(dhnd,'Geophysical Data',dhnd,ret)
call H5Aread_f(dhnd,H5T_NATIVE_REAL,geo1,sz1,ret)
call H5Dread_f(dhnd,H5S_ALL_F,H5S_ALL_F,geo1,sz1,ret,label1,label2)

113 C read layer 1
114 if(gid(30:32).ne.'SND')then
115 ! get scale
call H5Aopen_f(dhnd,'SCALE FACTOR',ahnd,ret)
call H5Aread_f(ahnd,H5T_NATIVE_REAL,sca,sz1,ret)
call H5Dread_f(dhnd,H5S_ALL_F,geo1,sz1,ret,label1,label2)
119 call H5Aclose_f(ahnd,ret)
120 ! read
sz1(1)=y
sz1(2)=x
call H5Dread_f(dhnd,geo1,H5T_NATIVE_REAL,geo1,sz1,ret,label1,label2)
125 if(ret.lt.0)then
126 ! H5Dread error:
127 write(*,'(a,a)')'H5Dread error: '
128 call exit(1)
129 endif
130 call H5Dclose_f(dhnd,ret)
131 ! change scale
132 do j=1,y
133 do i=1,x
134 if(geo1(i,j).gt.-32767)geo1(i,j)=geo1(i,j)*sca
136 enddo
137 enddo
138 endif

Scale handling is automatically applied when using AMTK. For details to P.22.
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 H5Open_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(i)
155         endif
156         call H5Dclose_f(dhnd)
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.

```
357 C memory free
358     deallocate(geo1)
359     if(gid(30:32).eq.'SND')then
360         deallocate(geo2)
361     endif
```

**Release the memory.**

**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.
```
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.

```bash
#!/bin/sh

# library directory
HDF5=/home/user1/util/hdf5_1.8.4-patch1
SZIP=/home/user1/util/szip_2.1

# compiler
fc=ifort

# source filename
fsrc="readL3G_hdf5.f"

# output filename
out=readL3G_hdf5_f

# library order
lib="-lhdf5hl_fortran -lhf5.hl -lhf5_fortran -lhdf5 -lsz -lz -lm"

# f compile
cmd="$fc -g $fsrc -o $out -I$HDF5/include -I$SZIP/include -L$HDF5/lib -L$SZIP/lib $lib"

# garbage
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 -lhf5.hl -lhf5_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 >
$ 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 “readL3G_hdf5_f” is shown as follows.

```
$ ./readL3G_hdf5_f GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000.h5
input file: GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000.h5
GranuleID(RETRIEVE FROM FILENAME): GW1AM2_20130200_01M_EQMA_L3SGCLWLA000000
grid size x: 1440
grid size y: 720
ASCII ART OF GEOPHYSICAL DATA LAYER #1 (X/20GRID Y/40GRID)
+------------------------------------------------------------------------+
|                                                                        |
[#]: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
```
AMSRR2 Product List

Dataset used in sample programs are shown in Table 10 to 15. Apart from these, more datasets are stored in AMSRR2 product. For details to “AMSRR2 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

<table>
<thead>
<tr>
<th>Data Name</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Time</td>
<td>R8</td>
<td>TM</td>
<td>AMTK_getScanTime()</td>
<td></td>
<td>SCAN</td>
</tr>
<tr>
<td>Latitude of Observation Point for 89A</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_89A</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Longitude of Observation Point for 89A</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_89B</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Longitude of Observation Point for 89B</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_89B</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>* Latitude (Low mean)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_LO</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Latitude (Low mean)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_06</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Latitude (6G)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_07</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Latitude (7G)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_10</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Latitude (10G)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_18</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Latitude (18G)</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_23</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (6.9GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB06H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (7.3GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB07H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (7.3GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB07V</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (10.7GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB10H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (10.7GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB10V</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (18.7GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB18H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (18.7GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB18V</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (23.8GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB23H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (23.8GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB23V</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (36.5GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB36H</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (36.5GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB36V</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (89.0GHz,A,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89AH</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (89.0GHz,A,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89AV</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (89.0GHz,B,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89BH</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Brightness Temperature (89.0GHz,B,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89BV</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Pixel Data Quality 6 to 36</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_PIX_QUAL_LO</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Pixel Data Quality 89</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_PIX_QUAL_HI</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Land_Ocean Flag 6 to 36</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_LOF_LO</td>
<td>243 x SCAN x 6</td>
</tr>
<tr>
<td>Land_Ocean Flag 89</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_LOF_HI</td>
<td>486 x SCAN x 2</td>
</tr>
<tr>
<td>Earth Incidence</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_EARTH_INC</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Earth Azimuth</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_EARTH_AZ</td>
<td>243 x SCAN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Datasets marked by * are not stored in HDF5</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Time</td>
<td>R8</td>
<td>TM</td>
<td>AMTK_getScanTime()</td>
<td></td>
<td>SCAN</td>
<td></td>
</tr>
<tr>
<td>Latitude of Observation Point for 89A</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_RS_89A</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Longitude of Observation Point for 89A</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_RS_89B</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Latitude of Observation Point for 89B</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_RS_89B</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>* Longitude (For low frequency data)</td>
<td></td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_RS_LO</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res06,6.9GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES06_TB06H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res06,6.9GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES06_TB06V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res06,7.3GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES06_TB07H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res06,7.3GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES06_TB07V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res10,10.7GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES10_TB10H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res10,10.7GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES10_TB10V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res23,18.7GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES23_TB18H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res23,18.7GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES23_TB18V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res23,23.8GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES23_TB23H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res23,23.8GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES23_TB23V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res36,36.5GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES36_TB36H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res36,36.5GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES36_TB36V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res36,89.0GHz,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES36_TB89H</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (res36,89.0GHz,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_RES36_TB89V</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (original,89GHz-A,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89AH</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (original,89GHz-A,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89AV</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (original,89GHz-B,H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89BH</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (original,89GHz-B,V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_TB89BV</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Pixel Data Quality 6 to 36</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_PIX_QUAL_LO</td>
<td>486 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Land_Ocean Flag 6 to 36</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_LOF_RES_LO</td>
<td>243 x SCAN x 4</td>
<td></td>
</tr>
<tr>
<td>Land_Ocean Flag 89</td>
<td>U1</td>
<td>I4</td>
<td>AMTK_get_SwathInt()</td>
<td>AM2_LOF_RES_HI</td>
<td>486 x SCAN x 2</td>
<td></td>
</tr>
<tr>
<td>Earth Incidence</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_EARTH_INC</td>
<td>243 x SCAN</td>
<td></td>
</tr>
<tr>
<td>Earth Azimuth</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_EARTH_AZ</td>
<td>243 x SCAN</td>
<td></td>
</tr>
</tbody>
</table>

### Table 12. L2 low resolution data used in the sample programs

<table>
<thead>
<tr>
<th>Data Name</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Time</td>
<td>R8</td>
<td>TM</td>
<td>AMTK_getScanTime()</td>
<td>SCAN</td>
<td></td>
</tr>
<tr>
<td>Latitude of Observation Point</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_L2_LO</td>
<td>243 x SCAN</td>
</tr>
<tr>
<td>Longitude of Observation Point</td>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geophysical Data</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_SWATH_GEOA (*1)</td>
<td>LAYER x 243 x SCAN</td>
</tr>
<tr>
<td>Pixel Data Quality</td>
<td>U1</td>
<td>U1</td>
<td>AMTK_get_SwathUChar()</td>
<td>AM2_PIX_QUAL</td>
<td>243 x SCAN x LAYER</td>
</tr>
</tbody>
</table>


(*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

<table>
<thead>
<tr>
<th>Data Name</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Time</td>
<td>R8</td>
<td>TM</td>
<td>AMTK_getScanTime()</td>
<td>SCAN</td>
<td></td>
</tr>
<tr>
<td>Latitude of Observation Point for 89A</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_L2_89A</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Longitude of Observation Point for 89A</td>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latitude of Observation Point for 89B</td>
<td>R4</td>
<td>LL</td>
<td>AMTK_getLatLon()</td>
<td>AM2_LATLON_L2_89B</td>
<td>486 x SCAN</td>
</tr>
<tr>
<td>Longitude of Observation Point for 89B</td>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geophysical Data for 89A</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_SWATHA_GEOA (*1)</td>
<td>LAYER x 486 x SCAN</td>
</tr>
<tr>
<td>Geophysical Data for 89B</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_get_SwathFloat()</td>
<td>AM2_SWATHB_GEOA (*2)</td>
<td>LAYER x 486 x SCAN</td>
</tr>
<tr>
<td>Pixel Data Quality for 89A</td>
<td>U1</td>
<td>U1</td>
<td>AMTK_get_SwathUChar()</td>
<td>AM2_PIX_QUAL_A</td>
<td>486 x SCAN x LAYER</td>
</tr>
<tr>
<td>Pixel Data Quality for 89B</td>
<td>U1</td>
<td>U1</td>
<td>AMTK_get_SwathUChar()</td>
<td>AM2_PIX_QUAL_B</td>
<td>486 x SCAN x LAYER</td>
</tr>
</tbody>
</table>


(*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

<table>
<thead>
<tr>
<th>Data Name</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brightness Temperature (H)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_getGridFloat()</td>
<td>AM2_GRID_TBH (*1)</td>
<td></td>
</tr>
<tr>
<td>Brightness Temperature (V)</td>
<td>U2</td>
<td>R4</td>
<td>AMTK_getGridFloat()</td>
<td>AM2_GRID_TBV (*1)</td>
<td></td>
</tr>
</tbody>
</table>


(*1) For details to P.13, Figure 6. L3 granule ID.

### Table 15. L3 data [Geophysical quantity] used in the sample programs

<table>
<thead>
<tr>
<th>Data Name</th>
<th>HDF5 Output</th>
<th>AMTK Output</th>
<th>AMTK Access Function</th>
<th>AMTK Access Label</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geophysical Data</td>
<td>I2</td>
<td>R4</td>
<td>AMTK_getGridFloat()</td>
<td>AM2_GRID_GEOA (*2)</td>
<td>(1) x LAYER</td>
</tr>
</tbody>
</table>


(*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

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of observation points for each scan of high resolution data.</td>
<td>AM2_DEF_SNUM_HI</td>
<td>486</td>
</tr>
<tr>
<td>Number of observation points for each scan of low resolution data.</td>
<td>AM2_DEF_SNUM_LO</td>
<td>243</td>
</tr>
<tr>
<td>2-byte signed integer loss value</td>
<td>AM2_DEF_IMISS</td>
<td>-32768</td>
</tr>
<tr>
<td>2-byte unsigned integer loss value</td>
<td>AM2_DEF_UIMISS</td>
<td>65535</td>
</tr>
<tr>
<td>1-byte unsigned integer loss value</td>
<td>AM2_DEF_CMISS</td>
<td>255</td>
</tr>
<tr>
<td>4-byte real loss value</td>
<td>AM2_DEF_RMISS</td>
<td>-9999.0</td>
</tr>
</tbody>
</table>

### Table 17. AMTK defined L3 parameters (number of pixels for each projection)

<table>
<thead>
<tr>
<th>Projection</th>
<th>Direction [Number]</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equi-rectangular projection (Low resolution)</td>
<td>Longitude</td>
<td>AM2_DEF_L3L_EQ_X</td>
<td>1440</td>
</tr>
<tr>
<td>Equi-rectangular projection (High resolution)</td>
<td>Longitude</td>
<td>AM2_DEF_L3H_EQ_X</td>
<td>3600</td>
</tr>
<tr>
<td>Equi-rectangular projection (High resolution)</td>
<td>Latitude</td>
<td>AM2_DEF_L3H_EQ_Y</td>
<td>1800</td>
</tr>
<tr>
<td>Northern polar stereo projection (Low resolution, TB/SIC)</td>
<td>Width</td>
<td>AM2_DEF_L3L_PN1_X</td>
<td>304</td>
</tr>
<tr>
<td>Northern polar stereo projection (High resolution, TB/SIC)</td>
<td>Length</td>
<td>AM2_DEF_L3L_PN1_Y</td>
<td>448</td>
</tr>
<tr>
<td>Southern polar stereo projection (Low resolution, TB/SIC)</td>
<td>Width</td>
<td>AM2_DEF_L3L_PS_X</td>
<td>316</td>
</tr>
<tr>
<td>Southern polar stereo projection (High resolution, TB/SIC)</td>
<td>Length</td>
<td>AM2_DEF_L3L_PS_Y</td>
<td>332</td>
</tr>
<tr>
<td>Southern polar stereo projection (High resolution, SND)</td>
<td>Width</td>
<td>AM2_DEF_L3H_PS_X</td>
<td>790</td>
</tr>
<tr>
<td>Southern polar stereo projection (High resolution, SND)</td>
<td>Length</td>
<td>AM2_DEF_L3H_PS_Y</td>
<td>830</td>
</tr>
<tr>
<td>Northern polar stereo projection (Low resolution, SND)</td>
<td>Width</td>
<td>AM2_DEF_L3L_PN2_X</td>
<td>432</td>
</tr>
<tr>
<td>Northern polar stereo projection (High resolution, SND)</td>
<td>Length</td>
<td>AM2_DEF_L3L_PN2_Y</td>
<td>574</td>
</tr>
<tr>
<td>Northern polar stereo projection (High resolution, SND)</td>
<td>Width</td>
<td>AM2_DEF_L3H_PN2_X</td>
<td>1080</td>
</tr>
<tr>
<td>Northern polar stereo projection (High resolution, SND)</td>
<td>Length</td>
<td>AM2_DEF_L3H_PN2_Y</td>
<td>1435</td>
</tr>
</tbody>
</table>