EUMETSAT

Object-Orientated IDL EPS Product Reader: User Manual

Doc.No. : EUM/OPS-EPS/MAN/08/0029
Issue : v3C
Date : 11 May 2015
WBS :

EUMETSAT
Eumetsat-Allee 1, D-64295 Darmstadt, Germany
Tel: +49 6151 807-7
Fax: +49 6151 807 555
http://www.eumetsat.int

Document Change Record

Issue / Revision	*Date*	*DCN No*	*Summary of Changes*
V1			Replaces document (EUM.EPS.SYS.UMN.04.003) which is now obsolete.
V2	17/06/08		Page 1 & 4 – Modified Chapters 1.1, 1.2, and 2 to include a description of the new procedural read routine described in Chapter 7. Page 2 & 3 – Updated list of references (RD28-RD31). Page 5 – Changed name of compressed file containing the software release to match the naming convention used on the EUMETSAT website. Page 7 – Added record_sphr method to ASCAT Class in Diagram 1. Page 10 – Updated Table 1, adding new PFS version numbers. Page 16 – Added Chapter 6.6 CROPPED PRODUCTS, that describes which cropped products produced by the UMARF can/cannot be read with the software. Page 17 – Added new chapter describing the procedural read routine. Page 20-21 – Updated Appendix tables with the new software added in version 1.1. Removed Chapter 7 on copyright and licence information. The EUMESAT website now describes the licence agreement for the software.
V2A	01/07/08		Corrected the page numbers stated above for changed pages/paragraphs for issue V2.
V2B/V2C	05/08/10		Updated Chapter 1.3, removing the date, revision and issue number of the reference documents. The new document management tool does not support the old numbering system. Added two new references for the ASCAT L2, and IASI PCC PFS documents. Added two new acronyms in Chapter 1.4 for SOMO (Soil Moisture), and PCC (Principle Component Compression). The software no longer supports pre-launch formats, so all references to CN26 and April04 baseline have been removed from the document.

Issue / Revision	*Date*	*DCN No*	*Summary of Changes*
			Updated Figure 1 to include the new ASCAT level 2 soil and moisture product, and the number of product subclasses is incremented by 1 in chapter 4.1 and 4.4. Table 1 modified to represent the product format major and minor versions supported by the software. The columns for the CN26 and April04 baseline have been removed. The template file directories supplied with the software have been merged into a single 'Templates' directory and the template filename no longer has the PFS revision and version number appended to the filename. The template filenames now include the format major version number. The Appendix has been updated to reflect these changes. The HTML directory has been removed and is no longer referenced in the Appendix. The software supports the new GOME-2 format that is currently not operational, Table 1 notes the new format is supported and the list of programs in the Appendix includes the new template file.
V2D	01/04/11		Added a reference for the AVHRR level 2 winds product in Chapter 1.3. Updated Figure 1 to include the new AVHRR level 2 winds product, and the number of product subclasses is incremented by 1 in chapter 4.1. and 4.4. The AVHRR level 2 winds product added to Table 1. The current format major version for GOME has been changed to 12 in the same table. The Appendix has been updated to include the AVHRR level 2 class and templates files. The records defined in the templates_avhrr_fmv_10.pro file have been corrected in the table to represent the level 1 A/B records.
V2E	16/06/11		Chapter 8 has been added, detailing the EUMETSAT copyright and Freeware licence agreement. The descriptions of each Chapter in the overview (Chapter 2) have been updated.
V2F	22/11/12		Updated Table 1 with the latest format major version of the ASCAT level 1 products. Incremented the version number of the software in Chapter 3 of the installation details. Added the templates_ascat_fmv_12.pro file to the list of program files in the Appendix.

Issue / Revision	*Date*	*DCN No*	*Summary of Changes*
			The overview in chapter 2 is amended to highlight the importance of using the PFS documentation when interpreting the EPS operational native product format. Changed formatting and layout of some tables to meet standards for OPS documentation. Compiled and added tables for figures and tables to meet layout requirements for OPS Documentation..
V2G	05/02/13		Updated Table 1 with the latest format major version of the AVHRR level 2 winds products. Added the templates_avhrr_l2_fmv_11.pro file to the list of program files in the Appendix.
V3A	13/02/13		Updated the document signature table. Updated the distribution list table, renaming the DM tool used for electronic distribution.
V3B	18/03/14		Updated the version of the software release and the version of IDL used to develop and test the software in chapter 3. Updated table 2 with the new format major version used for IASI level 2 products. Added new IASI level 2 templates file to the list of templates in the Appendix.
V3C	11/05/15		Corrected a spelling mistake in the Table of Tables. Corrected the links in chapter 5.5.5 to refer to 5.5.1 and 5.5.2, and not 6.5.1 and 6.5.2. Updated version of the software release in chapter 4. Updated table 2 with the new format major version used for ASCAT level 2 products, and added '(02/2D/2T)' in the AVHRR level 2 field to describe the different AVHRR level 2 wind products. Added new ASCAT level 2 templates file to the list of templates in the Appendix

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

1.1 Purpose and Scope

The purpose of this document is to outline the use of IDL and the objected-orientated EPS product readers to process and analyse the data contained within the level 1a, 1b, 1c, and level 2 EPS operational native products. In addition to the object-orientated designed read routine, a procedural designed read routine is included to read operational native products with fixed-size records.

1.2 Document Structure

No.	*Section Name*
1	Introduction
2	Overview
3	Installation Instructions
4	Class Design Overview
5	Program usage
6	Description of records, fields, data types and irregular arrays
7	Procedural IDL EPS Product Reader
8	Copyright and Licence information
9	Appendix

1.3 Reference Documents

RD 1	IDL Reference Guide	
RD 2	Generic Product Format Specification	EPS.GGS.SPE.96167
RD 3	AMSU-A Level 1 Product Format Specification	EPS.MIS.SPE.97228
RD 4	ASCAT Level 1 Product Format Specification	EPS.MIS.SPE.97233
RD 5	ASCAT Level 2 Soil Moisture Product Format	EUM/OPS-EPS/SPE/07/0343
RD 6	ATOVS Level 2 Product Format Specification	EPS.MIS.SPE.980759
RD 7	AVHRR Level 1 Product Format Specification	EPS.MIS.SPE.97231
RD 8	GOME Level 1 Product Format Specification	EPS.MIS.SPE.97232
RD 9	GRAS Level 1 Product Format Specification	EPS.MIS.SPE.97234
RD 10	HIRS Level 1 Product Format Specification	EPS.MIS.SPE.97230
RD 11	IASI Level 1 Product Format Specification	EUM/OPS/USR/06/1855
RD 12	IASI Level 1 PCC Product Format Specification	EUM/OPS-EPS/SPE/08/0195
RD 13	IASI Level 2 Product Format Specification	EUM/OPS-EPS/SPE/08/0338
RD 14	MHS Level 1 Product Format Specification	EPS.MIS.SPE.97229
RD 15	UMARF LEO Format Descriptions	EUM/OPS/USR/06/1855
RD 16	AVHRR Level 2 Polar Winds PFS	EUM/OPS-EPS/SPE/08/0338

1.4 Acronyms

Acronym	*Meaning*
AMSU-A	Advanced Microwave Sounding Unit-A
ASCAT	Advanced wind SCATterometer Acronym used as a simplified name for the AVHRR, MHS, AMSU-A and HIRS group of instruments
AVHRR	Advanced Very High Resolution Radiometer
CCB	Configuration Control Board
EPS	EUMETSAT Polar System
GEADR	Global External Auxiliary Data Record
GIADR	Global Internal Auxiliary Data Record
GOME-2	Global Ozone Monitoring Experiment-2
GPFS	Generic Product Format Specification
GPP	Ground Prototype Processor
GRAS	GPS Receiver for Atmospheric Sounding
GRH	Generic Record Header
HIRS	High-Resolution Infrared Radiation Sounder
IASI	Infrared Atmospheric Sounding Interferometer
IDL	Interactive Data Language
IPR	Internal Pointer Record
MDR	Measurement Data Record
MHS	Microwave Humidity Sounder
MPHR	Main Product Header Record
OO	Object-Orientated
PCC	Principle Component Compression
PFS	Product Format Specification
PGS	Product Generation Specification
PPF	Product Processing Facility
SOMO	Soil Moisture
SPHR	Specific Product Header Record
UMARF	Unified Meteorological Archive and Retrieval Facility
VEADR	Variable External Auxiliary Data Record
VIADR	Variable Internal Auxiliary Data Record

2 OVERVIEW

IDL, the Interactive Data Language software, is ideal for data analysis, visualisation and cross-platform application development. Many of the built-in features of IDL are designed specifically for remote sensing applications. IDL has been selected to read, process, and analyse the EPS operational native products.

The IDL EPS product readers are able to process multiple versions of the EPS operational native product formats that have been generated since the launch of Metop-A, and due to its object-orientated design makes them ideal for application development. A basic knowledge of object-oriented programming and IDL is therefore required to make full use of the software once the EPS products are read into memory.

The EPS operational native product formats are not self-descriptive; the user must therefore refer to the respective PFS (Product Format Specification) documentation in order to determine the contents of the product. The PFS document details the records in the product, and the fields in each record, including the field name, dimensions, units, data type, scaling factors, and a description. The user must therefore use the PFS documentation in conjunction with the IDL EPS product reader software.

An object-oriented approach has been taken to allow data management and format verification methods to be defined within the product class, and to allow data persistence. This guide is divided into eight chapters to help the reader install and run the software.

Chapter	*Contents and Instructions for...*
3	Outlines the installation instructions for Windows- and Unix-based OS, along with the minimum software requirements for this release.
4	Describes the OO class design and details some of the more useful methods.
5	Describes the calling function read_eps.pro that is used to create an instance of the product class, read the whole product and/or extract selected fields.
6	Provides a brief explanation of the naming conventions, data handling, and data storage used for both the record and field data read by the *readProduct* and *readField* methods respectively.
7	Describes the calling function read_fixed_eps.pro; this is a new procedural version of the read routine based on the software package ‘EPS IDL product reader version 2.0’.
8	Provides details of the EUMETSAT copyright and Freeware licence agreement that should be read before using the software.
9	The appendix including a list of all the programs supplied in this release.

3 INSTALLATION

3.1 Supplied Files

The EPS Product IDL Reader software can be found in the following compressed file:

zip_tools_eps_oo_idlreaders.tgz

The directory EPS_OO_IDL_V1_6_RELEASE is produced once the file is uncompressed. The source code can be found under the src directory (see the Appendix for a full file listing).

3.2 Supported Platforms and Requirements

The IDL EPS product reader software was developed and tested under Windows XP using IDL version 8.3. The software works on both big and little Endean machines and has been tested on UNIX and PC workstations. The source code is not compatible with versions of IDL older than 5.6.

3.3 Compilation

If your version of IDL is older than 5.6 some of the built-in IDL functions such as SWAP_ENDIAN_INPLACE will not be recognised and the software will fail. Therefore, the newer IDL routines will need to be replaced with older functions compatible with your version of IDL.

3.4 Unix Set-up

If IDL cannot locate the functions and procedures, the IDL_PATH system variables must be changed. Contact your system administrator or alternatively edit your start-up script using the set environment UNIX command ‘setenv’ as used in the following example:

setenv IDL_PATH +<location of IDL>: +<directory where software is stored>

If you already have the system variable $IDL_PATH set, simply append the top level directory to your list, remember to append ‘+’ to this directory so that IDL will search all subdirectories.

setenv IDL_PATH +/usr/local/rsi/idl/lib: +/EPS_OO_IDL_V1_6_RELEASE

Alternatively if you are using the idlde graphical interface to IDL set the IDL path from the File->Preferences menu (see Section 3.5 for more details).

3.5 Windows Set-up

If IDL cannot locate the functions and procedures, the IDL path needs to be expanded. The simplest way is to use the idlde graphical interface to IDL. Follow these instructions:

1. Chose **Preferences** from the **File** menu.
2. From the new window, choose **Path** and click on the **Insert** button.
3. Enter the location of the directory where the software is contained.
4. Make sure that both the *Enable Path Cache* box and the *New path* boxes are ticked.
5. Click **Apply.** This will automatically include all the subdirectories needed and save their location for later sessions.

4 CLASS DESIGN OVERVIEW

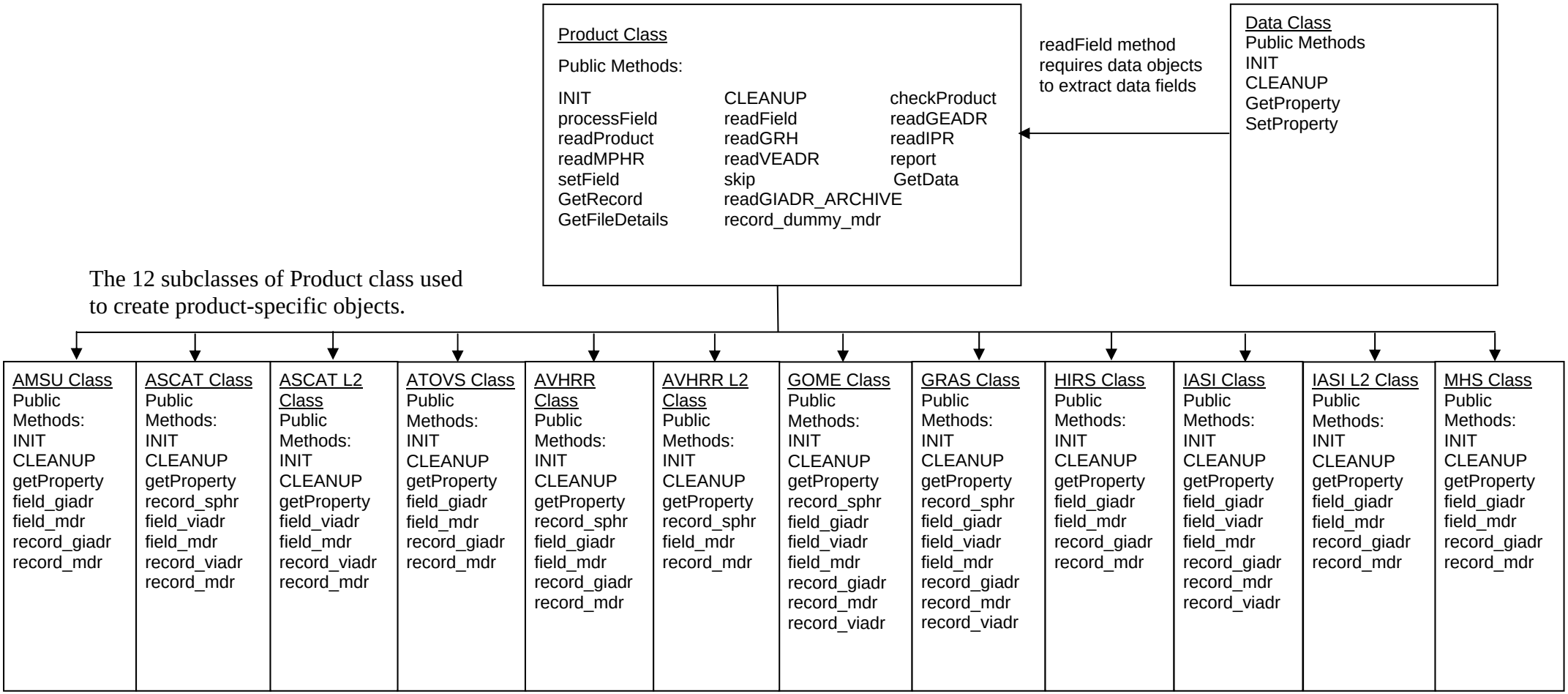

Figure 1: Class design flow chart

The IDL EPS product readers are made up of three main classes:

Product Class	Contains the attributes and methods generic to all EPS products
Product Subclasses (AMSU, ATOVS etc.)	Contain the attributes and read routines specific to the product type.
Data Class	Is a simple container encapsulating information regarding a field in a record. This information is used by the product class method readField to extract data from a record and store a pointer to the retrieved data in the data object.

Figure 1 represents the relationship of each of these classes and lists the methods for each class. From the diagram, the product subclasses inherit from the product class. The product class read methods use method overloading to call the appropriate read methods of the product subclass. The readField method requires a pointer to a data object array containing the details of the fields to be read.

4.1 The Product Class

The product class has 12 subclasses representing the level 1 and level 2 EPS products. Its purpose is to encapsulate the attributes and contain the reading methods generic to all EPS products. The two main methods of the product class are readProduct and readField. However, before trying to read the product it must first determine if it is a valid EPS product and if so what type of product i.e. AVHRR, AMSU, etc. The format version also needs to be determined before the appropriate record definitions are loaded. The product class method checkProduct performs all of these tests and must be called prior to running the main read routines (refer to *read_eps.pro* for an example of the checkProduct method).

4.2 ReadProduct

The readProduct method reads the entire product and stores pointers to the data records in the product object. It is possible to limit the range of MDR records read when initialising the product type object by using the *ystart* and *yend* keywords.

4.3 ReadField

The *readField* method only reads the field specified in the data objects. The data objects must therefore be created before calling this method. This method is very useful if memory allocation is an issue, since it only retains the data requested. Large fields such as SCENE_RADIANCE can be reduced into a single channel using indexing. If the requested data does not exist or the indexing is incorrect the readField function does not fail, but the data is not retrieved for that data object. The developer must therefore be careful to check that the DATA_VALUES pointer in the data object is valid. It is possible to limit the range of MDR records read when initialising the product type object by using the ystart and yend keywords.

Note: The *readField* method only extracts data from the GIADR, VIADR and MDR records and stores a pointer to this data in the data object (the IPR, GEADR and VEADR are ignored by this method). However the MPHR and SPHR are read by default and stored in the product object rather than the data objects.

4.4 The Product Subclasses

There are 12 product subclasses for the level 1 and level 2 EPS products (refer to Figure 1). Each class encapsulates the data and contains the reading methods specific to that product type; they inherit all the generic attributes and methods to read and validate an EPS product from the product super class. The records read using the *readProduct* method can be retrieved using the *GetProperty* and *GetRecord* methods, whilst data read using the *readField* method must use the *GetProperty* method of the data object.

4.5 The Data Class

The data class simply stores information regarding a field in a record, semantics data used for post-processing, and a pointer to the data in memory generated by the *readField* method. The field information is a pre-requisite for the *readField* method, so the following arguments are required during the initialisation of the data object.

Argument	***Definition***
NAME	A string containing a unique name used to identify the data object.
FIELD	A string containing the field name in a record.
INDEX	A string containing the index dimension. This allows you to subset data from a field. If you don't want to subset the field then enter '[*]'.
RECORD	A string containing the record number as stated in the relevant PFS.
SUBCLASS	A string containing the record subclass number as stated in the PFS

Table 1: Arguments in readField method

If the *readField* method is successful it stores a pointer to the data in the DATA_VALUES attribute in the data object. If the method is unsuccessful, the program does not fail but the DATA_VALUES attribute will contain a null pointer. The readField method also generates a list of record start and stop times taken from the GRH of the records in which the fields reside, and stores a pointer to this data in the data object. Finally, the data object also has a number of other attributes which can be set using the *setProperty* method, namely SF and UNITS. These are not pre-requisite for the read routines but will be useful for post-processing purposes since the product does not contain this information.

5 PROGRAM

This section describes the purpose and usage of the calling function read_eps.pro. This function creates an instance of a product object and calls the *checkProduct* method to determine the product type, format version and processing level, before generating the specific product subclass object.

5.1 Purpose

To read EPS products and encapsulate pointers to the data read into product objects for analysis and application development.

5.2 Inputs

Product	*Level*	Format of Major. Minor Version	
		Launch	*Current*
AMSU-A	1A/B	10.0	10.0 [RD3]
ASCAT	1A/B	10.0	12.0 [RD4]
ASCAT	2 SOMO	10.0	12.0 [RD5]
ATOVS	2	10.0	10.1 [RD6]
AVHRR	1A/B	10.0	10.0 [RD7]
AVHRR	2 (02/2D/2T)	10.0	11.0 [RD16]
GOME	1A/B	10.0	12.0 [RD8]
GRAS	1A/B	10.0	10.0 [RD9]
HIRS	1A/B	10.0	10.0 [RD10]
IASI	1A/B/C	10.0	11.0 [RD11]
IASI	1 PCC	10.0	10.0 [RD12]
IASI	2	10.0	11.0 [RD13]
MHS	1A/B	10.0	10.0 [RD14]

Table 2: EPS products, as defined in their PFS

5.3 Outputs

IDL structures read into memory.

- EPS product object

5.4 Syntax

Result=read_eps(Filename [,FIELD=field][,/PRODUCT][,ystart=value][,yend=value])

Keywords – PFS format: [,FIELD=FIELD] [,/PRODUCT]

Keywords – Tools: [,ystart=*value*] [,yend=*value*]

5.4.1 Result

The function returns a zero if it fails or the product object if it succeeds.

5.4.2 Arguments

Filename – A string containing the full path and filename of the EPS product to be read.

5.4.3 Keywords

FIELDS – Set this keyword to a pointer to one or more data objects. The *readField* method of the product class uses the product information in the data object to extract the data for the specified field in the EPS product.

PRODUCT – Setting this keyword causes read_eps.pro to call the *readProduct* method of the product class. This method reads all of the records in the product (if ystart and/or yend is set then the range of MDR records will be read). This method stores pointers to the record data in the product object.

ystart – Allows the user to specify the first MDR to start reading (0-based index).

yend – Allows the user to specify the last MDR to read (0-based index).

5.5 Examples

The following examples have been added to demonstrate the use of the calling function read_eps.pro and the product class read methods, namely *readProduct* and *readField*.

5.5.1 ReadProduct

The following example is used to read in an AVHRR level 1 EPS product:

```
prod=read_eps(Filename, /PRODUCT)
```

Running this command reads the AVHRR product specified in the Filename argument and stores a pointer to each record read in the product object returned.

5.5.2 ReadField

The following example would be used to read channel 1 scene radiance data from an AVHRR level 1 EPS product:

```
data=OBJ_NEW('data', 'Channel 1', 'SCENE_RADIANCES', '[*, 0]','8','2')

prod=read_eps(filename, FIELDS=PTR_NEW(data))
```

Since *ystart* and *yend* keywords are not set, the entire channel 1 data are read.

5.5.3 Data Retrieval from a Product Object

The following example demonstrates how data is retrieved from the product object generated in example 5.5.1. In this example the channel 1 data are extracted from the SCENE_RADIANCES field in the MDR records.

Retrieve the pointer to the array of pointers for each MDR record, stored in the ‘mdr’ attribute.

```
prod->GetProperty,MDR=mdr
```

Create a container for the channel 1 data.

```
ch1=intarr(2048,N_ELEMENTS(*mdr))
```

De-reference the MDR records and extract the channel 1 data from the scene_radiance field.

```
for i=0L,N_ELEMENTS(*mdr)-1 do ch1(*,i)=(*((*mdr)(i))).SCENE_RADIANCES(*,0)
```

Display the channel 1 data (ch1) using the IDL function TVSCL.

```
TVSCL,ch1
```

Always remember to destroy the product object and all heap memory when finished.

```
OBJ_DESTROY,prod
```

5.5.4 Data Retrieval from a Data Object

The following example demonstrates how data are retrieved from the product object generated in example 5.5.2.

Before the data can be retrieved the correct data objects in the product object need to be established. In this case there is only one data object but in the following example we do not make this assumption.

```
Prod->GetData, DATA = odata
for i=0, N_ELEMENTS(*odata) –1 do begin
    (*odata)(i)->GetProperty,NAME=name
    if(name eq ‘Channel 1’) then $
        (*odata)(i)-> GetProperty,DATA_VALUES=data_values
endfor
```

Once the correct data object is established the data_values pointer is used to retrieve the channel 1 data. The data does not need to be extracted from SCENE_RADIANCES since the readField method has already performed this task for you.

Create a container for the channel 1 data.

```
ch1=intarr(2048, N_ELEMENTS(*data_values))
```

De-reference the channel 1 data.

```
for i=0L,N_ELEMENTS(*data_values)-1 do ch1(*,i)=*((*data_values)(i))
```

Display the channel 1 data (ch1) using the IDL function TVSCL.

```
TVSCL,ch1
```

Always remember to destroy the product object and all heap memory when finished.

```
OBJ_DESTROY,prod
```

Channel 1 (ch1) contains the same data as in the previous example, but has been extracted from the field pointer stored in the data object rather than the record pointer stored in the product object. The advantage of using this method rather than readProduct is that only the channel 1 scene radiance is stored in memory and not the whole product. This is extremely important when memory allocation is an issue.

5.5.5 Retrieving the MPHR from the Product Object

Records such as the MPHR, IPR, GEADR and VEADR are automatically read when either the readProduct or readField methods are called; the product class therefore has a method to retrieve these records, namely GetRecord. The following example demonstrates how to retrieve the MPHR and display information regarding the record using the IDL help command combined with the structure keyword; this is a continuation from the examples shown in either 5.5.1 or 5.5.2.

Retrieve the MPHR record from the product object.

```
prod->GetRecord,MPHR=mphr
```

Display contents of structure.

```
help,(*mphr), /st
```

6 RECORDS, FIELDS, DATA TYPES, AND IRREGULAR ARRAYS

This chapter has been included to provide the user with some prior knowledge of the format of the product record structures, record and field naming conventions, and storage design.

6.1 Records

Each record is stored in memory and a product attribute is assigned a pointer to this data. If more than one record of the same subclass type exists then an array of pointers is generated and a pointer to this array is stored in the product attribute. In both cases the attribute has the same name as the record it represents i.e. mphr, sphr, ipr, geadr, giadr, veadr, viadr or mdr. For products with multiple subclasses of the same record type the subclass number is appended to the attribute name i.e. ‘viadr1’,’viadr2’…’viadr*n*’. To avoid keyword ambiguity in GRAS product attributes, zero plus the subclass number (for subclass numbers less than 10) is appended to the attribute name, e.g. ‘mdr01’. These pointers are set in the product object when the readProduct method is called. Using pointers rather than fixed sized structures for variable sized records/fields in products such as ATOVS Level 2, IASI Level 2, GOME and GRAS, greatly reduces the amount of memory required to store the product.

6.2 Fields

The field names (tag names) in the record structure follow the same naming convention as defined in the product’s PFS. The only exception to this rule is when the name is not a valid tag name for an IDL structure–names containing white space, symbols, numbers at the beginning of the name, and/or duplicate names within the same structure. These tag names are replaced with valid IDL tag names i.e. white space is replaced with an underscore, symbols are removed, ‘NUM_’ is prepended to names beginning with a number, and duplicate tag names have the occurrence number appended to the tag name making them unique.

6.3 Data Types

The EPS products have a variety of different data types, most of which are represented in IDL i.e. 16 bit integers (int), 32 bit integers (long), 64 bit integers (long64), bytes, and strings. However, there are no specific IDL data types that represent vintegers, Boolean, enumerators, time and date structures, compound data structures, or bit strings. These data types are treated as either binary arrays or read into predefined IDL structures.

6.4 Irregular Arrays

IDL is an array-orientated language that can process regular arrays extremely efficiently; however irregular arrays are not supported in IDL. Many fields in various EPS products have irregular arrays, such as the covariance matrix in the ATOVS and IASI Level 2 products, or the band data in the GOME-2 MDR records. Since only a few fields are variable in length for the above products, these fields contain pointers to the variable sized array data in memory. However, almost every field in a GRAS product can be variable in length, and creating pointers for these fields is not practical. The GRAS read routine reads multi-dimensional irregular arrays into a one-dimensional array. This solves the problem, but in order to access specific slices from the array an accessor function needs to be developed (no such routine exists in this release).

6.5 Dummy MDR

In previous versions of this software dummy MDR records are treated as blank MDR records. This is memory intensive and could cause some confusion, since an MDR filled with zeros could be interpreted as a real MDR. In this release a pointer to the dummy MDR records read are stored in the ‘dummy_mdr’ attribute within the product object.

6.6 Cropped Products

The EUMETSAT Data Centre (formerly UMARF) allows the user to order cropped products using the record start/stop time and/or band number. The IDL readers can read products that have been modified using the record start and stop time, but does not support products cropped by band number since this modifies the format of the product.

7 PROCEDURAL IDL EPS PRODUCT READER

The object-orientated IDL EPS product readers are designed to read EPS products with variable sized records. Since IDL cannot store an array of variable-sized structures, an array of pointers is created with each element pointing to a record structure. In order to develop a generic read routine for all EPS operational native products, every record is assumed to be variable in size and is assigned a pointer.

Although the object-orientated IDL EPS product reader is better equipped to read all EPS operational native products and includes methods to free allocated memory, extract selected fields from a record, and read multiple formats of a product, many users still prefer to use the older procedural version of the software. This is mainly because the user needs to call specific methods to access a record, rather than returning the record when setting the appropriate keyword in the procedural read routine. Furthermore, once the pointer to the record/records is returned the user needs to de-reference the pointer that could contain an array of pointers requiring further processing. This can become time consuming and confusing especially when working from the IDL prompt. For this reason a read routine based on the procedural 'IDL EPS product reader version 2' has been included in this release. The read routine only works with products with fixed-size records; therefore they cannot read GOME-2, GRAS, ATOVS level 2 or IASI level 2 products. As mentioned above, the object-orientated version of the read routine is designed specifically to handle variable sized records, and there would be no benefit to the user if they are included in the procedural version of the read routine, i.e. the user would still need to de-reference an array of pointers to access the records.

Including a procedural version of the read routine in the object-orientated IDL EPS product reader release has the benefit that both read routines share the same repository of product format templates. The procedural read routine can therefore read multiple format versions of a product. Another new feature of the procedural read routine is the ability to read cropped products produced by the EUMETSAT Data Centre.

7.1 Purpose

To read EPS products into IDL structures for testing, visualisation and analysis.

7.2 Inputs

All input products listed in Table 2 with the exception of GOME-2, GRAS, ATOVS level 2 and IASI level 2.

7.3 Outputs

IDL structures read into memory.

- MPHR (optional)
- SPHR (optional)
- IPR (optional)
- GIADR (optional)
- GEADR (optional)
- VIADR (optional)
- VEADR (optional)
- MDR (optional)

7.4 Syntax

read_fixed_eps, Filename

Keywords – PFS format: [,MPHR=MPHR] [,SPHR=SPHR] [,IPR=IPR] [,GIADR=GIADR] [GEADR=GEADR] [,VIADR=VIADR] [,VEADR=VEADR] [,MDR=MDR]

Keywords – Tools: [,ystart=*value*] [,yend=*value*] [,/HALT] [,/REPORT] [,sys_err=sys_err] [,prod_err=prod_err] [,err_message=err_message]

7.4.1 Arguments

Filename – The full path and filename of the EPS product to be read.

7.4.2 Keywords

Key	*Description*
MPHR	Structure containing the MPHR record.
SPHR	Structure containing the SPHR record.
IPR	Array of structures containing the IPR records.
GIADR	Structure containing each GIADR subclass record.
GEADR	Structure containing each GEADR subclass record.
VIADR	Structure containing an array of VIADR records for each subclass.
VEADR	Structure containing an array of VEADR records for each subclass.
MDR	Array of structures containing the MDR records. (Alternatively the range of MDRs if ystart and yend are specified.) Dummy MDR records are ignored; to determine if any dummy MDR records are present the user should use the REPORT keyword. To determine when dummy MDR records occur in the product the user should use the record start and stop times in the MDR generic record header.
ystart	Allows the user to specify the first MDR to start reading (0-based index).
yend	Allows the user to specify the last MDR to read (0-based index).
HALT	In conjunction with yend, HALT causes the program to stop reading the rest of the product once the last MDR specified by yend has been read. This keyword is useful for stopping the program at a specified MDR.
REPORT	Prints out the record type, subclass and the record size specified in the GRH for each record. The current file pointer for the beginning and end of each record is also printed out to the screen (the difference of these two pointers must match the record size in the GRH otherwise the program will break).
sys_err	This keyword is used to report any errors encountered during the runtime of the program, namely out-of-memory allocation errors, or any errors not caught by the prod_err keyword (see below) i.e. if the product is not compliant with the PFS. A value other than zero indicates an error was detected.
prod_err	This keyword is used to report any errors encountered whilst reading the product, i.e. invalid EPS product or errors in the record. A value of one indicates an error was detected.
err_message	This keyword is used to capture the details of the error from either the prod_err or sys_err keywords. The error messages from prod_err are hard coded in the program e.g. ‘INVALID EPS PRODUCT – INVALID RECORD CLASS’, whilst the error messages from sys_err are produced by IDL and captured in the !ERR_STRING.

7.4.3 Example

The following example would be used to read in an AVHRR level 1 EPS product:

```
read_fixed_eps, Filename, MPHR, SPHR=SPHR, IPR=IPR, GRH=GRH, GIADR=GIADR, GEADR=GEADR, MDR=MDR, ystart=0, yend=99
```

Running this command on the IDL prompt will read the AVHRR product specified in the filename argument and generate the specified structures into memory. The MDR structure will contain the total number of real MDR records stored within the first 100 MDR records; if no dummy MDR records are present this will be 100 records.

8 COPYRIGHT AND LICENCE INFORMATION

The ‘Object-Orientated IDL EPS Product Reader’ software is copyright to EUMETSAT, and is distributed under the terms and conditions of the EUMETSAT Freeware licence agreement. Please read the following licence information carefully and adhere to the conditions before using the software provided.

Freeware Licence Disclaimer:

“The user acknowledges and shall at all times respect EUMETSAT’s intellectual property rights in the software. EUMETSAT shall at all times retain such intellectual property rights in the software and in all copies thereof regardless of form.

The user may freely use, copy, modify and/or distribute the software. The user agrees when using the software in any recognisable form to name EUMETSAT as the source by including "©(year) EUMETSAT". Furthermore, in any distributed version of the software, including modifications or derivatives, the present licence text shall be included.

The user of the software may provide feedback, report problems and suggest enhancements to the software to EUMETSAT. In addition, the user shall grant to EUMETSAT unrestricted use of this information.

Neither EUMETSAT nor its Member States are liable for the usefulness or proper functioning of software, nor do they accept any liability for any consequences, whether direct or indirect, of any use of software or for any results related to the use of software or for any right or claims by third parties related to all or any part of software or its use. Where the source code is made available to users, this is done without any warranty, and EUMETSAT will not provide any support for its use and customisation.”

9 APPENDIX

Inside the Object-Orientated IDL EPS Product Reader software package, the code is stored in the src directory containing all the procedures/functions and record structure definitions. The remainder of this appendix lists the full contents of the base directory and sub-directories.

9.1 File listing of base directory

File Name	*Usage*
README.txt	Description of the product formats compatible with this release.

9.2 File listing of src/ directory

File Name	*Usage*
read_eps.pro	Creates a product type object.
read_fixed_eps.pro	Procedural designed read routine.
read_mphr.pro	Defines/reads ASCII MPHR into a structure.
read_grh.pro	Defines GRH structure.
data_types__define.pro	Definition of generic data types.
product_class__define.pro	Product class definition.
amsu_class__define.pro	AMSU-A L1A/B class definition.
ascat_class__define.pro	ASCAT L1A/B class definition.
ascat_l2_class__define.pro	ASCAT L2 SOMO class definition.
atovs_class__define.pro	ATOVS L2 class definition.
avhrr_class__define.pro	AVHRR L1A/B class definition.
avhrr_l2_class__define.pro	AVHRR L2 class definition.
gome_class__define.pro	GOME L1A/B class definition.
gras_class__define.pro	GRAS L1A/B class definition.
hirs_vers__define.pro	HIRS L1A/B class definition.
iasi_class__define.pro	IASI L1A/B/C class definition.
iasi_l2_class_define.pro	IASI L2 class definition.
mhs_class__define.pro	MHS L1A/B class definition.
data_class__define.pro	Data class definition.
resolve_all_programs.pro	A compilation program for debugging purposes ONLY (the .sav file generated should not be used).

9.3 File listing of src/Templates/ directory

File Name	*Usage (Template/Read)*
templates_amsu_fmv_10.pro	Defines all AMSU-A L1A/B records.
templates_ascat_fmv_10.pro	Defines all ASCAT L1A/B records.
templates_ascat_fmv_11.pro	Defines all ASCAT L1A/B records.
templates_ascat_fmv_12.pro	Defines all ASCAT L1A/B records.
templates_ascat_l2_fmv_10.pro	Defines all ASCAT L2 SOMO records.
templates_ascat_l2_fmv_11.pro	Defines all ASCAT L2 SOMO records.
templates_ascat_l2_fmv_12.pro	Defines all ASCAT L2 SOMO records.
templates_atovs_fmv_10.pro	Defines/reads all ATOVS L2 records.
templates_atovs_fmv_10_1.pro	Defines/reads all ATOVS L2 records.
templates_avhrr_fmv_10.pro	Defines all AVHRR L1A/B records.
templates_avhrr_l2_fmv_10.pro	Defines all AVHRR L2 records.
templates_avhrr_l2_fmv_11.pro	Defines all AVHRR L2 records.
templates_gome_fmv_10.pro	Defines/reads all GOME L1A/B records.
templates_gome_fmv_11.pro	Defines/reads all GOME L1A/B records.
templates_gome_fmv_12.pro	Defines/reads all GOME L1A/B records.
templates_gras_fmv_10.pro	Defines/reads all GRAS L1A/B records.
templates_hirs_fmv_10.pro	Defines all HIRS L1A/B records.
templates_iasi_fmv_10.pro	Defines all IASI L1A/B/C/PCC records.
templates_iasi_fmv_11.pro	Defines all IASI L1A/B/C records.
templates_iasi_l2_fmv_10.pro	Defines/reads all IASI L2 records.
templates_iasi_l2_fmv_11.pro	Defines/reads all IASI L2 records.
templates_mhs_fmv_10.pro	Defines all MHS L1A/B records.