Consultative Committee for Space Data Systems

REPORT CONCERNING SPACE DATA SYSTEM STANDARDS

STANDARD FORMATTED DATA UNITS — A TUTORIAL

CCSDS 621.0-G-1

GREEN BOOK

May 1992

AUTHORITY

Issue:	Green Book, Issue 1
Date:	May 1992
Location:	CCSDS Panel 2 Meeting, May 1992, Oberpfaffenhofen, Germany

This report reflects the consensus technical understanding of the Panel 2 members representing the following member Agencies of the Consultative Committee for Space Data Systems (CCSDS):

- British National Space Centre (BNSC) / United Kingdom
- Canadian Space Agency (CSA) / Canada
- Centre National D'Etudes Spatiales (CNES) / France
- Deutsche Forschungsanstalt für Luft und Raumfahrt (DLR) / FRG
- European Space Agency (ESA) / Europe
- Instituto de Pesquisas Espaciais (INPE) / Brazil
- National Aeronautics and Space Administration (NASA) / USA
- National Space Development Agency of Japan (NASDA) / Japan

The following observer Agencies also technically concur with this report:

- Department of Communication/Communications Research Centre (DOC/CRC) / Canada
- Institute for Space Astronautics and Science (ISAS) / Japan

This Report is published and maintained by:

CCSDS Secretariat
Communications and Data Systems Division, (Code-OS)
National Aeronautics and Space Administration
Washington, DC 20546, USA

DOCUMENT CONTROL

Document	***Title***	***Date***	***Status/ Remarks***
CCSDS 621.0-G-1	Report Concerning Space Data System Standards: Standard Formatted Data Units -- A Tutorial, Green Book, Issue 1	May 1992	Issue 1 (this document supersedes CCSDS 620.0-G-1)

CONTENTS

Figures

REFERENCES

[1] "Recommendation for Space Data System Standards: Standard Formatted Data Units -- Structure and Construction Rules", CCSDS 620.0-B-1, Blue Book, Issue 1, Consultative Committee for Space Data Systems, May 1992 or later.

[2] "Recommendation for Space Data System Standards: Parameter Value Language Specification (CCSD0006)", CCSDS 641.0-B-1, Blue Book, Issue 1, Consultative Committee for Space Data Systems, May 1992 or later.

[3] "Recommendation for Space Data System Standards: Standard Formatted Data Units -- Control Authority Procedures", CCSDS 630.0-R-2, Red Book, Issue 2, Consultative Committee for Space Data Systems, April 1992 or later.

[4] "Recommendation for Space Data System Standards: ASCII Encoded English (CCSD0002)", CCSDS 643.0-R-1, Red Book, Issue 1, Consultative Committee for Space Data Systems, May 1992 or later.

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to explain the rationale of the Standard Formatted Data Unit (SFDU) concept and outline the Structure and Construction Rules with the help of examples.

This document serves as a companion book to Reference [1], which gives a formal description of the SFDU Structure and Construction Rules.

1.2 Document Organisation

This document is a tutorial. It is structured as follows:

- Section 2 presents the basic problems of data interchange and introduces the concept used to address them.
- Sections 3 describes SFDU structuring and the basic structural building blocks.
- Section 4 describes the methods of packaging data units within SFDU products.
- Section 5 explains the techniques for delimiting data units.
- Sections 6.1 and 6.2 describe the two main structural units which can be used in SFDU products, the Information Data Unit (Section 6.1) and the Description Data Unit (Section 6.2).
- Section 7 states all the requirements which the SFDU concept has to satisfy and gives a rationale for each requirement. It also provides a compliance table to demonstrate the fulfilment of all the requirements.
- After descriptions of all aspects of the SFDU concept, Section 8 gives a number of examples from real projects, which show how the SFDU is used in practice.
- Annexes A and B present a complete summary of the acronyms and the terminology used in this document;
- An index is supplied covering all the major terms in the document; in this the first page referenced by the index points to the definition of the term.

In this report structure diagrams are used to explain some of the structures presented. The following conventions are used in these diagrams:

- The item named to the left of the **:=** symbol is the item being defined;
- The diagram on the right of the **:=** symbol is the definition;
- A vertical branch point represents a choice;

- A repetition is indicated by a loop back covering the object to be repeated. If there are the symbols **n<x** next to the loop back, then it means the loop can be repeated only zero to **x** number of times;

- The termination of each structure is represented by an **o** symbol.

For example:

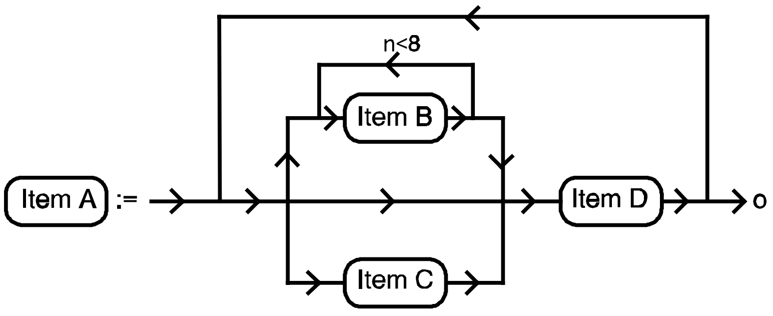


Figure 1-1: Example Structure Diagram

In this example Item A is defined as:

A sequence of two items:

- The first is a choice between Item B and Item C or nothing.

- The second is one Item D.

If selected, Item B may be present from one to eight times.

The whole structure described may be repeated any number of times until the path to the **o** symbol is selected.

Of course if any items on the right (B, C or D) contain an Item A, the definition is recursive.

The following names used in this document are registered trade marks: VAX, VMS, DECNET, UNIX, MSDOS.

2 BACKGROUND

2.1 The Problems of Data Interchange and Archiving

Information interchange is defined as the transfer of information on a system at one location to a system at another location. In this report "information" means any type of knowledge that can be exchanged and "data" refers to the representative forms of that "knowledge". In some instances, for example, in mission operations, the systems interchanging information (e.g., the experiment on-board a spacecraft and the experimenter at his laboratory or institute) in the interchange process have a common view of the data. The problems of information interchange become primarily those associated with identifying the source of the information (e.g., a particular experiment), and understanding any additional information (e.g., quality control) that may have been added to the original data as it flowed from source to destination.

In other instances, for example, in retrieving data from an archive, the source and destination systems (e.g., a researcher's system and the data archive) may never have shared a common view of the data. In such cases it is imperative that a common view be established between them. The problems which arise are:

- Determining the information location (where it resides) and its identification;
- Accessing the data (i.e., transferring it from its source);
- Interpreting or understanding the data.

The interchange of science information is particularly subject to these problems because the data is long-lived, may have many diverse users, may require supplementary data to understand or process it in a meaningful sense and may have relationships with other data. Furthermore, information interchange often takes place between different types of computers with different operating systems each of which may have, for example, different data representation conventions; in such cases it is said that the computer environment is heterogeneous.

At present, a typical data management and processing system is dedicated to a particular flight mission or project. Data is acquired, processed, shared to some level with other members of the investigating team, and eventually archived. Such dedicated systems are usually implemented specially for a project's requirements. Further, documentation of the data and its formats may not be complete and up-to-date since the investigators already have an understanding of their data which is kept current by actively working with the data, maintenance of the processing software, etc.

This practice results in two major problems:

- A new information interchange data system must be implemented for each mission. This is expensive.
- As a result of inadequate or out-of-date documentation, the reuse of data at some future period by a researcher, who was not a part of the original investigating team, will be made difficult or even impossible. Inadequate documentation can also be a problem for the original investigators if they do not work with the data for a period of time.

There is a trend for space and earth scientists to move away from traditional mission-oriented research activity towards inter-disciplinary investigations. This requires the integration of information from many sources. Thus data interchange requirements are becoming more demanding. Furthermore, data rates and volumes are also increasing, adding further importance to the need for efficient and economical information interchange systems.

2.2 Overview of the Standard Formatted Data Unit (SFDU) Concept

The Standard Formatted Data Unit (SFDU) concept provides standardised techniques for the automated packaging and interpreting of data products. It puts no constraint on the format of the user data, and can thus accommodate standard formats developed by other organisations or user communities. It operates in a heterogeneous environment.

The SFDU concept has been developed to address the problems discussed in Section 2.1. It offers the following:

- A low overhead, internationally recognised data labelling scheme which permits self-identification of data objects;
- Standard techniques for providing complete and unambiguous data descriptions;
- Procedures for registration and administration of these data descriptions;
- Techniques for packaging labelled data objects into larger data products;
- Sufficient standardisation to allow the development of generic software to support the retrieval, access, parsing and presentation of SFDU data objects, while allowing those objects to have individual formats to satisfy particular application and user needs.

The CCSDS has developed Recommendations which address these aspects and they are listed on the Reference page of this report. This report primarily addresses the data labelling and data object packaging which form the subject of the Recommendation on Structure and Construction Rules (Reference [1]). The Structure and Construction Rules of Reference [1] govern the basic interchange format of information in the SFDU domain. The examples in Section 8 show how the first four aspects outlined above have been automated in practical applications. These examples, which are from real projects, use software and thus demonstrate the feasibility of the fifth aspect; generic software.

2.3 General SFDU Usage

It is useful to consider two typical methods of using SFDUs. The first is ad-hoc data interchange, in which human intervention and interpretation is the prime characteristic. In this case it is sufficient to package data arbitrarily. The second is automated operational data interchange, in which most human involvement has happened at the time an information system was being designed and implemented. In this case, system preparation will involve interface negotiation, design and possibly the establishment of local conventions, e.g. communication protocols, media types, etc.

2.3.1 An Example of Ad-Hoc Data Interchange

A scientist is asked to send some time series data to another scientist, along with ancillary and catalog data. The scientist uses various resources to decide how to do so, including this book and a generic SFDU designer/generator.

The receiving scientist applies an SFDU parser to discover the structure of the received product. Having some knowledge of the use of the classes and the meaning of hierarchical structures, the scientist can gain an understanding of the data. For example, a catalogue attribute object has been packaged with its associated image.

In this example, understanding has been aided by the structure of the SFDU, as interpreted by a human.

2.3.2 An Example of Operational Data Interchange

For automated exchange of data products, we assume that the human is not in the loop except in an operational role (computer operations, software support, quality control etc.). In this case the data producers and receivers must agree on a number of things beforehand.

As an example of production usage, consider an automated catalogue and active archive system. Several data producers have received telemetry data, and have done whatever is necessary to generate their science products, which have been designed in advance. During the operational phases of the mission, product generation software is in place which generally runs automatically, usually with some support from human operators. Each of the data producers have their own data requirements, which are likely to be qualitatively and quantitatively different. The receiving active archive, on the other hand, will require certain common elements, perhaps including global attribute data for each product, and local attribute data for each data object catalogued. Based on these requirements, some general guidelines or rules are developed for the data products; the producers are expected to adhere to the guidelines. They may be very tight or very loose, but the agreement amounts to a local standard or convention.

The guidelines will need to include the allowable "shape" of the SFDU (e.g., if a single catalogue attribute object must precede a number of application (primary data) data objects, does it apply to the first or to all of the application data objects?, or are related data objects packaged together?, or are they put in separate products?).

The guidelines must define the allowable contents of the objects within the LVOs of the SFDUs and the semantics of the contents. There may be a range of data types available, or they may be strictly limited. For example, a time series may be represented in double precision floating point or integer form. One might allow either, if distinguishing information is somehow conveyed within the SFDU. (The data object identifier (ADID) might serve, or one could use attributes in a catalogue attribute object. In the latter case, the attribute must be known to the receiver.) As another example, if the catalog objects are to be specified in a formal language such as PVL, the languages to be supported must be agreed upon. Finally, it might be necessary to specify Data Entity Dictionaries to specify the meaning of each data element.

The matters which must be addressed are summarized as follows:

- The organisation of the data objects within the SFDU product: what is their order, how are they nested, which are optional, which are related, etc?.

- The contents of each data object.

Furthermore, before a receiver of a product can even begin to access the data, he must know the type of medium/operating system used, and how to read the initial SFDU label. Once this header is read he can navigate through the product, but an initial "sticky label" is required on the medium to get started.

If it is not possible to obtain agreements from receivers as well as producers, as in the case of data producers who do not know all of their future customers, the producer must carefully make decisions and document them, so that all of the above information is available.

The CCSDS Panel 2 is considering means of standardising or providing registration for specifications that address the above issues.

It is noted that the steps described above have always been necessary; the agreements on media, product structure, data object content and format are normally documented in an Interface Control Document, signed off by the product generating authority on the one hand, and the product receivers on the other.

The SFDU concept provides automatable techniques for designing structure products, for labelling its components and providing data descriptions, thus removing the heavy reliance of traditional systems on paper documentation.

3 THE SFDU BASIC ELEMENTS AND SFDU STRUCTURING

This section describes the techniques that are used to build SFDUs. This includes the basic building block, the higher level packaging structures and the methods of labelling and identifying each component of the SFDU structure.

3.1 SFDU Building Blocks - The Label-Value-Object

The basic SFDU building block is comprised of a LABEL field and a VALUE field, and is referred to as a Label-Value-Object (LVO). This structure is the fundamental structural element used to build SFDUs. The LVOs themselves are made up of a sequence of octets.

In the SFDU approach, data exchanged between open (independent) data systems are tagged with a LABEL, as shown in Figure 3-1.

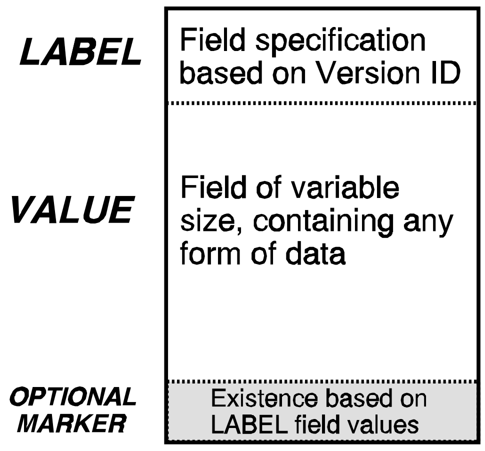

LABEL
Field specification based on Version ID
VALUE
Field of variable size, containing any form of data
OPTIONAL MARKER
Existence based on LABEL field values

Figure 3-1: LABEL-VALUE Encoding Structure

The LABEL contains the following sub-fields:

- An identifier of the format and meanings of all the other LABEL sub-fields;
- An identifier of the description of the format and meaning of the data in the VALUE field;
- An identifier that gives an indication of the type of data in the VALUE field;
- The necessary information required to delimit the VALUE field.

A complete definition of the sub-fields of the LABEL is given in Section 3.4.

The VALUE field may contain any form of data that can be described by a user defined data description or by a CCSDS recognised data description. The method used to delimit this field, and a description of the data in this field, are identified through the associated LABEL sub-field.

The optional marker field is required by some delimitation techniques to delimit the VALUE field (See

Section 5.2).

3.2 SFDU Structuring

SFDU data products are constructed from the basic LVO in one of two ways. If the VALUE field of the LVO contains purely user data it is termed a "Simple LVO". If, on the other hand, the VALUE field of the LVO contains purely LVOs, it is termed a "Compound LVO".

SFDU products are always packaged in a special kind of Compound LVO called the Exchange Data Unit (EDU). Only EDUs may be interchanged between systems. Special types of Compound LVOs are also provided to package together application data (the Application Data Unit (ADU)) and data description data (the Description Data Unit (DDU)).

The CCSDS defined categories of Simple and Compound LVOs, which vary depending upon the type of data or LVOs respectively that they contain, are detailed in the following sections.

3.2.1 Simple LVOs

Data in a Simple LVO may be viewed as belonging to one of the following categories:

- Application data; that is the data which is of primary interest (typically measurements or data derived from measurements);
- Supplementary data; that is data that is considered to enhance the understanding of the associated data;
- Data description information, telling how the application data are formatted, including such details as size of the data fields, numerical or other representations used and the meanings of the fields;
- Data cataloguing and/or data production information, giving certain overall attributes of the data, for example, date of generation, instrument used, instrument location, general information about the way the data was collected, relayed or processed, etc .

Any one of these types of data may be contained in the VALUE field of a single LVO. The structure of a Simple LVO is described in Figure 3-2.

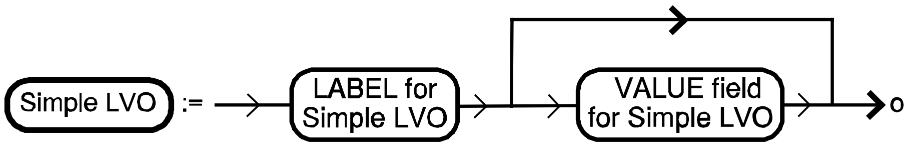


Figure 3-2: Structure Diagram of a Simple LVO

3.2.2 Compound LVOs

Compound LVOs are LVOs which contain within their VALUE field a sequence of one or more LVOs, each of which can be a Simple or Compound LVO itself. LVOs that are contained in the VALUE field of a Compound LVO are deemed to be one "Structure Level" lower than that of the containing Compound LVO. If any of these contained LVOs are themselves a Compound LVO then they will themselves contain a sequence of LVOs; this sequence is at the next lower "Structure Level". This process may continue indefinitely leading to a succession of structure levels. This process is the way in which LVOs are nested. There are no rules dictating the number or order of Compound and Simple LVOs within a data product, except that there must be at least one Simple LVO at the lowest structure level of any Compound LVO (i.e., a Compound LVO cannot have a VALUE field of zero length). The structure of a Compound LVO can be described by Figure 3-3.

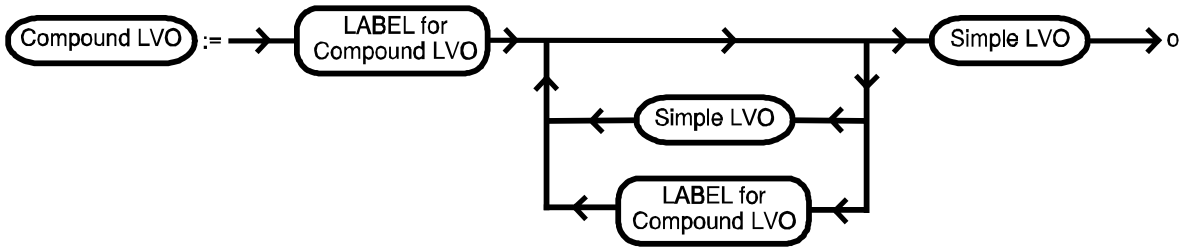


Figure 3-3: Structure Diagram of a Compound LVO

Figure 3-4 shows two examples of packaging LVOs within the VALUE field of LVOs.

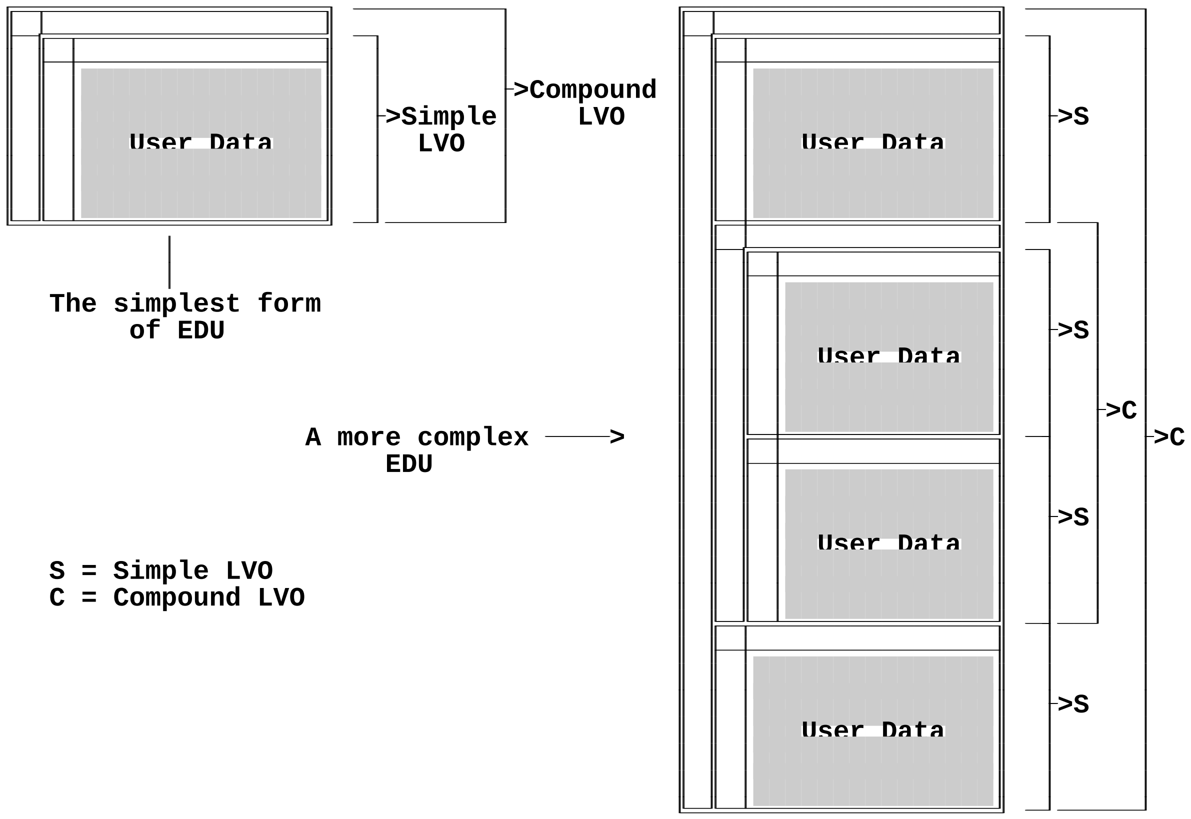


Figure 3-4: Examples of Packaging LVOs within LVOs

There are three types of Compound LVOs; there is the Exchange Data Unit (EDU), and two particular structures which must be packaged within an EDU. These are the Application Data Unit (ADU), which explicitly does not contain any data description information, and the Description Data Unit (DDU), which can contain only data description information.

3.2.2.1 Exchange Data Unit (EDU)

Typically an SFDU data product consists of not only the data *(e.g., an image, a set of measurement samples)*, but also all the supporting metadata that is needed to understand the data product. Any type of data may be contained within an Exchange Data Unit (EDU). SFDU data MUST be exchanged in the form of an EDU.

For example, the following could be packaged within an EDU:

- Information on the production of the product;
- Catalogue data pertaining to the product *(e.g., platform ID, data type, time span)*;
- A number of data instances comprising of application data and supplementary data *(e.g., images and image coordinates)*;
- A number of data descriptions that describe each of the application data formats and the supplementary data formats.

3.2.2.2 Application Data Unit (ADU)

The purpose of an ADU is to package application data instances *(e.g., measurement samples)* together with any necessary ancillary data *(e.g., sampling rate)* and identification data *(e.g., catalogue information)*, and to explicitly exclude any data description information. Typically an ADU will be used to "subset" a set of application data within an EDU. Software may be designed so that data stored within an ADU is directed towards one particular processing task, which selects the ADUs out of the data structure and skips over any others. Several such units typically appear within a data product. An ADU must always be packaged within an EDU. An example of the use of ADUs is shown in Figure 3-5.

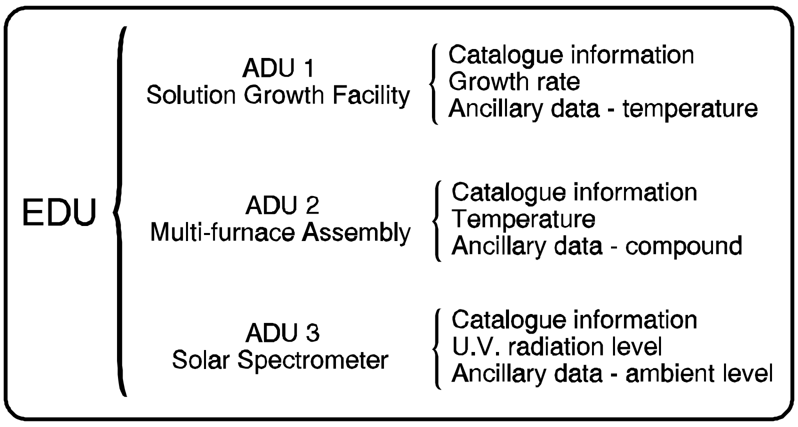


Figure 3-5: Example of the Use of ADUs

3.2.2.3 Description Data Unit (DDU)

A Description Data Unit is characterised as follows:

- It carries the description of a data object (typically syntactic information such as the format of a sample, and semantic information such as the name and units of the components of the sample);

- It explicitly links the data description to the data object to which it applies;

- It does not include any application data instances.

Like an ADU, a DDU must always be packaged within an EDU. Figure 3-6 shows a DDU that could be used to describe the data within the ADU shown in Figure 3-5. This data description data can be packaged together with the application data or held separately.

The underlying idea of the DDU is that a user system receiving a DDU knows that it only contains data descriptions, so that it can be directly routed to, for example, a processor for data descriptions. Equally the user system knows it need not look for application data in a DDU.

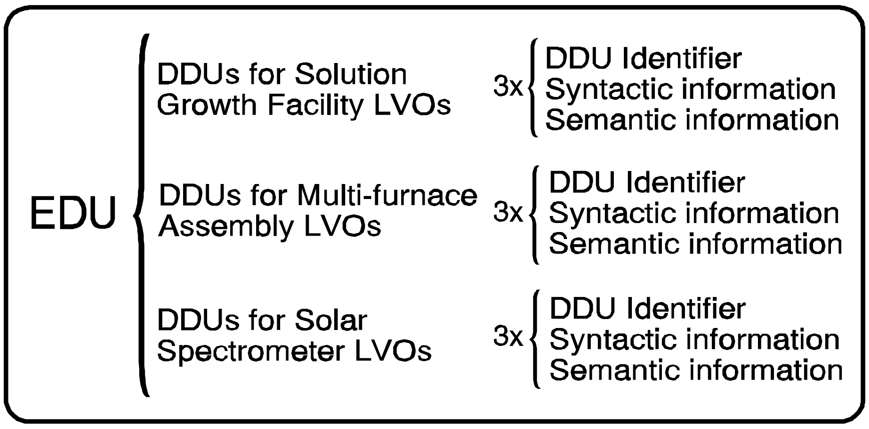
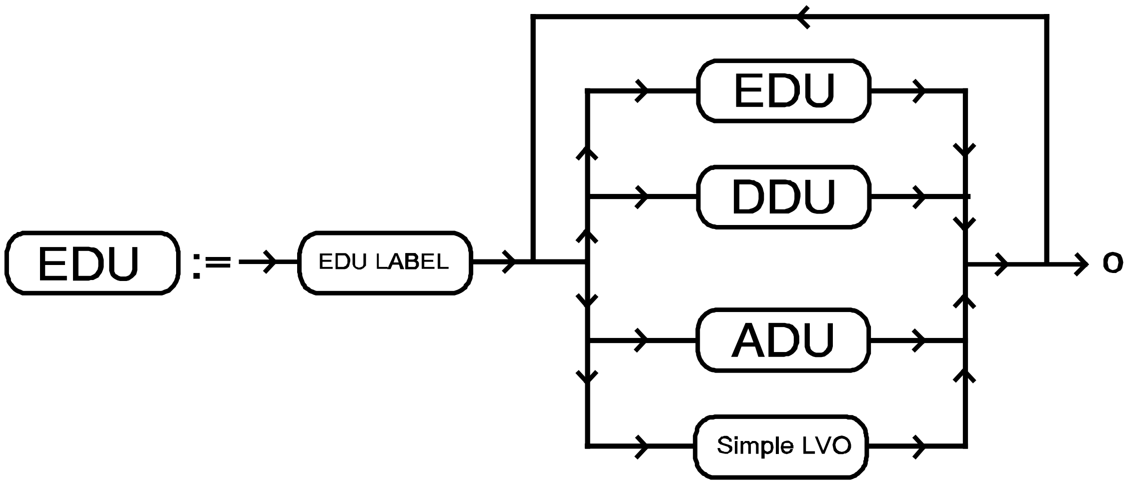


Figure 3-6: Example of the Use of DDUs

3.3 EDU Structure Diagram

The four structures that have been illustrated in the previous section are the Simple Label-Value-Object (LVO), the Exchange Data Unit (EDU), the Application Data Unit (ADU) and the Description Data Unit (DDU). These structures may be packaged together as indicated in the structure diagram of Figure 3-7 (overleaf). Not all the components on the right of the **:=** have to be included in all EDUs, but at least one Simple LVO must be present. The packaging is hierarchical with the highest level object being an EDU.



where: EDU = an Exchange Data Unit
ADU = an Application Data Unit
DDU = a Description Data Unit
and a Simple LVO is an LVO that has no further LVOs in its VALUE field

Figure 3-7: Structure Diagram of an EDU

3.4 LVO LABEL FIELD SPECIFICATIONS USED IN SFDUS

This section specifies the three versions of the LVO LABEL defined in the current SFDU Recommendation (Reference [1]) relating to the SFDU Structure and Construction Rules. The version of the LABEL is indicated by octet 4 of the LABEL in all cases. This sub-field of the LABEL is called the **Version ID**. A more complete definition of all the sub-fields appears below.

(An SFDU product must be immediately identifiable as falling within the CCSDS/SFDU domain; therefore octets 0 to 3 must contain the string **CCSD***, which is the Control Authority Identifier (CAID) for the CCSDS. Therefore, even though the Version ID dictates the format of the LABEL, it may not appear prior to the string* **CCSD** *and thus appears in octet 4.)*

The VALUE field may contain any form of data that can be described by a data description. The extent of this field, and a description of the data in this field, are identified through the associated SFDU LABEL.

Only Version IDs = "**1**", "**2**" and "**3**" (hereafter referred to as Versions 1, 2 and 3 respectively) are defined in the current Recommendation (Reference [1]). These correspond to a 20 octet LABEL of the general form shown in Figure 3-8.

CAID (ADID 1 of 2)	Version ID	Class ID	Delim. ID	Spare	DDID (ADID 2 of 2)	Delimitation Parameter	LABEL sub-field
0 - 3	4	5	6	7	8 - 11	12 - 19	Octet number

Figure 3-8: CCSDS LABEL Specification - Version ID = 1, 2 and 3

For Version IDs = 1, 2 and 3 the characters of the Restricted ASCII (RA) set are the only ones permitted in the sub-fields of the first 12 octets of the LABEL (Restricted ASCII is all the uppercase letters (A-Z), and the numbers (0-9) from the ASCII character set, see Reference [1], Annex D for a definition of the decimal codes).

For all three Version IDs, the following sub-fields have common meaning and values.

- **Control Authority Identifier (CAID)**: The Control Authority Identifier contains the identifier of the organisation that has assigned the DDID to the information describing the VALUE field. This Control Authority Office has the responsibility for maintaining this data description information and supplying it to user. The CAID is the first part of the ADID.

- **Data Description Identifier (DDID)**: The Data Description Identifier contains the identifier of the data description information held at the Control Authority Office, as identified by the CAID. The DDID is the other part of the ADID.

- The combination of the CAID and the DDID is called the **Authority and Description Identifier (ADID)**. The ADID uniquely identifies the data description information that applies to the associated VALUE field.

- **Class ID**: The Class ID indicates the kind of data contained in the VALUE field following the LABEL. The Class ID must be selected from those approved by the CCSDS. A list of approved Class IDs is found in Section 3.4.1.

- **Spare**: This is a spare octet which is set to the RA numeric character **`0`** (zero).

An overview of the sub-fields which change between the different versions is given in Table 3-1.

Version ID	Delimitation ID	Delimitation Parameter Meaning	Delimitation Parameter Representation
1	0 (zero)	Length	8 octet RA decimal integer
2	0 (zero)	Length	8 octet 64 bit binary integer
3	A	Length	8 octet RA decimal integer
	B	Length	8 octet 64 bit binary integer
	S	Marker completion pattern	8 printable ASCII characters
	C	Number of contiguous EOFs	8 octet RA decimal integer
	E	Number of sequential EOFs	8 octet RA decimal integer
	F	No delimitation parameter required	Fixed 8 octet RA string **`00000001`**

Table 3-1: Summary of the Version ID Dependent Sub-fields in the LABEL

The value contained within the Delimitation Parameter sub-field is used to complete the delimitation technique.

3.4.1 Overview of Class Identifiers

This section summarises the meaning of the Class IDs which may be used in all versions of LVO LABELs. The Class ID provides a general indication of the type of the data contained in the LVO VALUE field. The full description of the format of the VALUE field is given in the data description accessed via the ADID.

The Class IDs can be split into three basic categories, as shown in Figure 3-9. The Structure Classes handle the packaging of LVOs, the Service Classes provide CCSDS service mechanisms and the Data Classes contain the actual user data.

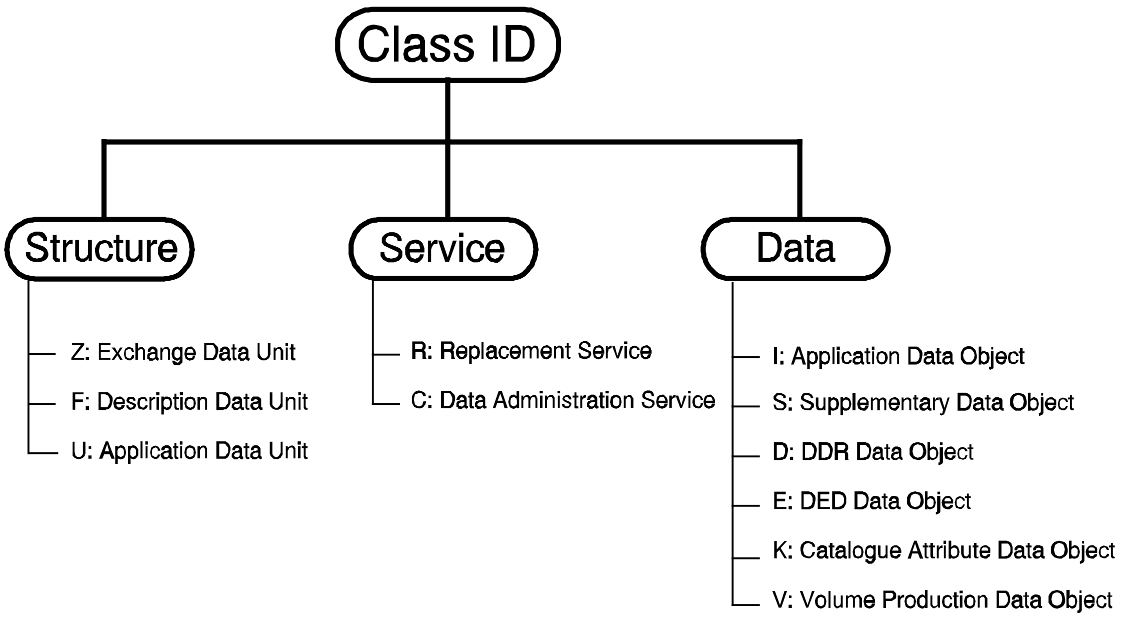


Figure 3-9: SFDU Class ID Breakdown

An application can assign further meanings to some of the classes. For example, the application may decide that it will distinguish between scientific and housekeeping data by packaging the former inside LVOs with Class ID = I and the latter inside LVOs with Class ID = S.

In addition certain LVOs, such as those with Class ID = K, relate to other LVOs. For example, a class K LVO may contain the title, author and date of publication of the LVO(s) following with Class ID = I. The scope of this association is left as an application decision (e.g., does a class K LVO at the highest level of a product apply to all the included data objects or only a subset?).

All such application specific matters must be fully described if the LVOs exchanged are to be properly interpreted. These descriptions must be available to all users of the LVOs concerned (as described in Section 6.2).

3.4.2 Overview of CCSDS Defined ADIDs

There are two types of permitted ADIDs:

1. Those defined by the CCSDS, appearing in CCSDS Recommendations, and beginning with the four RA characters **CCSD**. These are referred to as CCSDS ADIDs. When using a Structure or Service Class ID, a CCSDS ADID must be used whenever possible in order to maximise universal access to the corresponding LVOs;

2. Those defined by SFDU users, typically through data description registration with a Control Authority Office (See Reference [3]). These are referred to as User ADIDs.

While it is expected that most ADIDs will be assigned by Control Authority Offices to user-generated data descriptions, there are a number of standard descriptions that have been defined by the CCSDS for general use. These descriptions are assigned CCSDS ADIDs and they appear in approved CCSDS Recommendations. The CCSDS ADIDs currently identified as being associated with the structure and construction rules are defined in Reference [1] and are summarised in Table 3-2. The CCSDS ADIDs that appear to be missing from the table (i.e., CCSD0002, CCSD0006, CCSD0007 and CCSD0008) can either be found in other CCSDS Recommendations or correspond to specifications that are under development.

ADID	Value Field Definition
CCSD0001	VALUE field contains one or more LVOs
CCSD0003	VALUE field contains several "parameter=value" statements that optionally label external data objects before logically including them in the current structure
CCSD0004	VALUE field contains one or more "parameter=value" statements that identify a data description package and optionally reference other metadata objects
CCSD0005	VALUE field contains one or more LVOs, forming a Description Data Unit (DDU)
CCSD0009	VALUE field contains one or more LVOs, forming an Application Data Unit (ADU)

Table 3-2: Summary of CCSDS Defined ADID Specifications

3.5 Combining ADIDs and Class IDs

All LVO labels, by definition, include an ADID and Class ID combination. The purpose of this section is to summarise the possible combinations.

There are two categories of permitted ADID and Class ID combinations, these are defined as follows:

1. **CCSDS defined usage**: These combinations occur only with CCSDS ADIDs and correspond to cases where a particular CCSDS ADID specification has been specifically designed by the CCSDS to support a given Class ID. Where they exist such combinations are the preferred form;

2. **User defined usage**: These combinations occur only with User ADIDs, for which the User ADID must support the CCSDS defined usage of the Class ID.

Any other combinations of ADIDs and Class IDs are not permitted.

Table 3-3 summarises the ADID and Class ID combinations. It should be pointed out that the only reason for using a User ADID with a Structure or Service Class ID would be when the existing CCSDS ADIDs for these Class IDs do not meet the user's particular needs. In such a case, it is clear that these needs will not be supported through immediate universal recognition by the standard parsing tools.

ADIDs	**Class IDs**										
	Z	**F**	**U**	**R**	**C**	**I**	**S**	**D**	**E**	**K**	**V**
CCSD0001	✓✓	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
CCSD0002 †	▨	▨	▨	▨	▨	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
CCSD0003	▨	▨	▨	✓✓	▨	▨	▨	▨	▨	▨	▨
CCSD0004	▨	▨	▨	▨	✓✓	▨	▨	▨	▨	▨	▨
CCSD0005	▨	✓✓	▨	▨	▨	▨	▨	▨	▨	▨	▨
CCSD0006 ‡	▨	▨	▨	▨	▨	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
CCSD0009	▨	▨	✓✓	▨	▨	▨	▨	▨	▨	▨	▨
User defined ADIDs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

† English Text, see Reference [4]	✓✓	=	Permitted
‡ PVL, see Reference [2]	✓	=	Permitted if the ADID identified data description does not contradict the Class ID identified functionality
	▨	=	Not permitted

Table 3-3: ADID and Class ID Combination Categorisations

To summarise, Table 3-4 (overleaf) lists the CCSDS defined combinations of ADIDs and Class IDs that are important to the structure and construction of SFDUs as defined in Reference [1]. The fact that there is currently only one CCSDS ADID for each of the Structure and Service Classes is purely a state of the current Recommendation. More ADIDs may be developed for use with these Class IDs as and when the requirements arise for different implementations of the Class ID specified functionality.

Class ID	ADID	Permitted Instance of:
Z	CCSD0001	Exchange Data Unit (EDU)
F	CCSD0005	Description Data Unit (DDU)
U	CCSD0009	Application Data Unit (ADU)
R	CCSD0003	Replacement Service
C	CCSD0004	Data Administration Service

Table 3-4: CCSDS Defined Combinations of Class IDs and ADIDs

Note: In order to be a legal SFDU data product, the first label of any SFDU data products must have a Class ID = Z and an ADID = CCSD0001.

4 PACKAGING TECHNIQUES

Note: In the following diagrams the LVOs are shown in a stylised form, with only the information necessary to understand the concepts being presented. For example, in Figure 4-1, only the CAID, Class ID and DDID are indicated in the LABELs (L), although in reality all the other sub-fields of the LABELs are present.

4.1 Envelope Packaging

The idea is shown in Figure 4-1, which shows the simplest case, in which an EDU contains one LVO of application data, which could, for example, be a major frame of telemetry data. A more elaborate example, showing the envelope packaging of a "bundle" of telemetry minor frames is shown in Figure 4-2. Minor frames are bundled into an EDU for delivery of a partial major frame, so users can begin processing before the complete major frame is available.

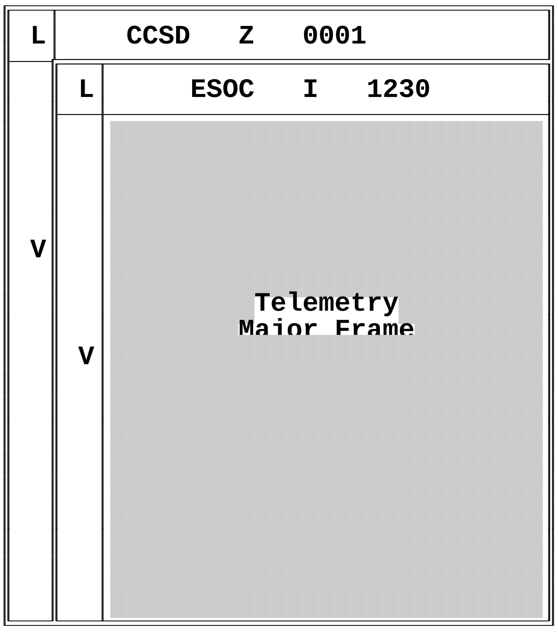


Figure 4-1: Envelope Packaging

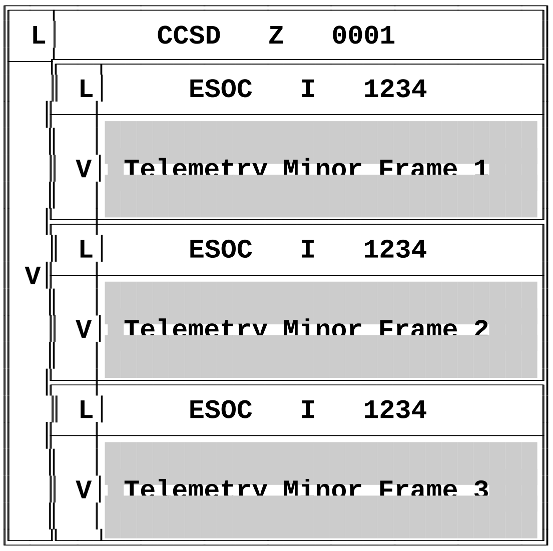


Figure 4-2: More Complex Envelope Packaging

4.2 Referencing Techniques - the Replacement Service

A referencing technique is provided to include data objects in a product even if those units are not stored contiguously with the rest of the product. This technique is called the "Replacement Service", because it allows logical inclusion of physically separate data objects (e.g., files) into SFDU products. The technique uses a class R LVO (the referencing LVO). This contains the reference information in its VALUE field in the form of Parameter Value Language statements (Reference [2]). The general concept is shown in Figure 4-3 (overleaf), where the class R LVO is envelope packaged in an EDU. It references a file called **`CCSDIMG`** containing an LVO with image data in its VALUE field.

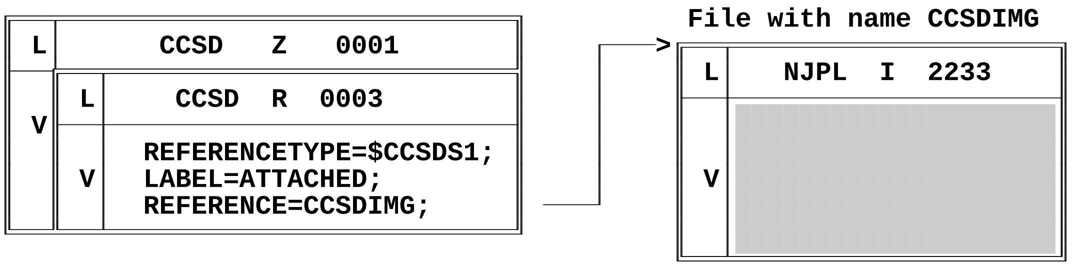


Figure 4-3: The Replacement Service - Referencing a File Containing a LVO

A more complicated case is shown in Figure 4-4 in which:

- The data in the referenced file is unlabelled and thus a LABEL has to be provided by the referencing data object (the class R LVO);
- Other LVOs are packaged together with the class R LVO to form quite a complex product;
- The provisional referencing environment **$CCSDS1** is being used. **$CCSDS1** is a basic **filename.ext** filenaming convention, with **filename** limited to a maximum of 8 characters and **ext** limited to a maximum of 3 characters. (For further discussion of referencing environments see Section 4.2.1.1).
- The LABEL supplied by the **LABEL** statement specifies that the ADID = ESOC1122, Class ID = I and Delimitation ID = F (Shared EOF).

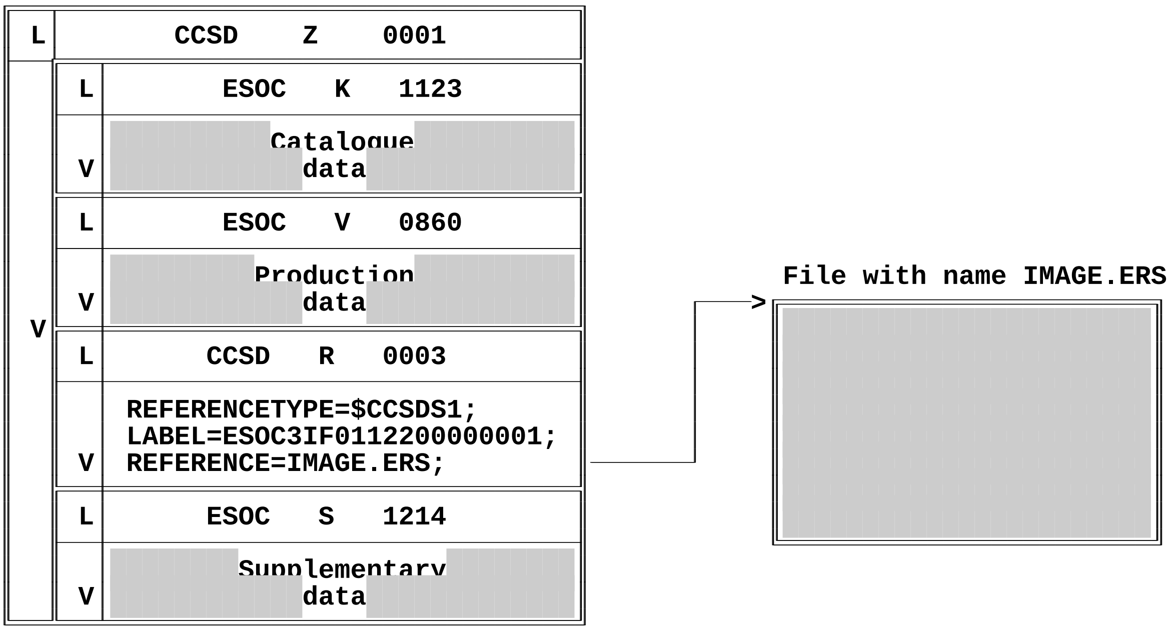


Figure 4-4: The Replacement Service - Referencing an Unlabelled Data Object in a File

The class R LVO has a CCSDS defined ADID = CCSD0003. The CCSD0003 specification describes a Simple LVO VALUE field for specifying parameters for use by the CCSDS replacement services provided by a class R LVO (the Referencing LVO). The Parameter Value Language, PVL (See Reference [2]), is used to specify the necessary parameters.

Three parameters (**REFERENCETYPE**, **LABEL** and **REFERENCE**) are specified by this ADID. They perform the following three tasks:

1. Define the name (**REFERENCETYPE**) of a referencing environment that is to be used to access external data object(s) from the Referencing LVO, for example, the filenaming convention and the type of data storage medium;

2. Provide a means to optionally add an LVO LABEL (**LABEL**) to external data object(s);

3. Provide a pointer (**REFERENCE**) to the start of external data object(s) within the specified referencing environment.

The three PVL statements that correspond to these three tasks are described below. The permitted ordering of these three statements in any one VALUE field is defined by Figure 4-5.

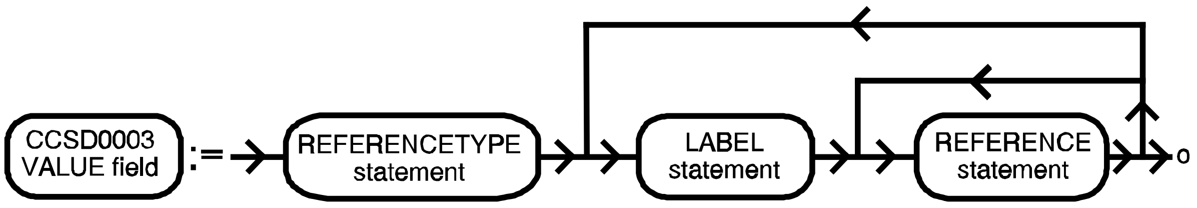


Figure 4-5: Structure Diagram of the PVL Statements within an LVO with ADID = CCSD0003

In the statement descriptions below any parameter names or values shown in upper case are keywords, whilst lower case is used to indicate user specified values. All keywords in a CCSD0003 VALUE field must be expressed in upper case.

a. **`REFERENCETYPE=refenv;`**
The **`refenv`** value names the referencing environment to be used. If **`refenv`** begins with the **`$`** character then it names a provisional CCSDS defined referencing environment (Section 4.2.1.1 describes provisional CCSDS defined referencing environments), otherwise it names an approved CCSDS defined referencing environment.

b. **`LABEL=string;`**
The **`string`** value allows an LVO LABEL field to be logically added to the beginning of the external data object(s). If **`string`** is the keyword **`ATTACHED`** then it means the external data object(s) shall have an LVO LABEL at the beginning and do not require a further LABEL. Otherwise, **`string`** shall conform to an LVO LABEL specification that is composed of printable ASCII characters (decimal codes 32 to 126).

c. **`REFERENCE=(name_1, name_2, . . . name_n);`**
Each **`name_x`** specifies the beginning of one or more external data objects within the defined referencing environment. The parentheses are optional if there is only one **`name_x.`**

The following should be noted concerning Figure 4-5:

i. There shall be one, and only one, **REFERENCETYPE** statement. This shall be the first statement;

ii. There shall be at least one **REFERENCE** statement;

iii. There shall be one **LABEL** statement before the first **REFERENCE** statement;

iv. There shall be no more than one **LABEL** statement between any two consecutive **REFERENCE** statements.

v. A **LABEL** statement can apply to many **REFERENCE** statements. The most recent **LABEL** statement will be used;

vi. The last statement in a VALUE field shall be a **REFERENCE** statement.

The following rules apply:

A. External data objects are referenced in the order of the **REFERENCE** statements. If a **REFERENCE** statement specifies more than one external data object, then these external data objects are referenced in the order that they appear in the **REFERENCE** statement.

B. If a **REFERENCE** statement refers to more than one external data object, then the given LABEL applies to each of these external data objects.

C. Each external data object terminates when the VALUE field delimited by the first LABEL (whether attached or supplied by a **LABEL** statement) has been completed.

D. Each external data object shall comply with the structure rules applying at the point where the external data object was referenced. Figure 4-6, shows a DDU which contains the mandatory class C LVO and then a class R LVO. The class R LVO references two files, **FILE1.DAT** that contains a labelled class D LVO (a syntax description) and **FILE2.DAT** that contain a labelled class E LVO (a semantic description). The two LVOs are logically included in the DDU thus forming a legal EDU structure.

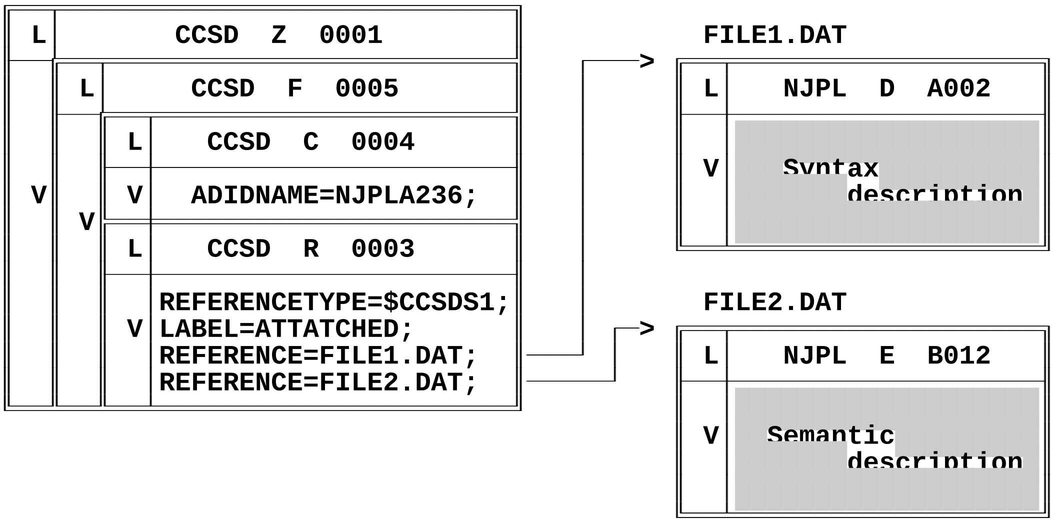


Figure 4-6: Following the Structure Rules when Referencing LVOs

Meanwhile Figure 4-7, shows another DDU with a similar structure. The mandatory class C LVO is present and then also the class R LVO. The class R LVO references two files, **`FILE1.DAT`** that contains a labelled class D LVO (a syntax description) and **`FILE2.DAT`** that contain a labelled class I LVO (application data). The class D LVO can be legally included, but it is illegal to have class I LVOs (application data) within a DDU, therefore the EDU structure shown is illegal

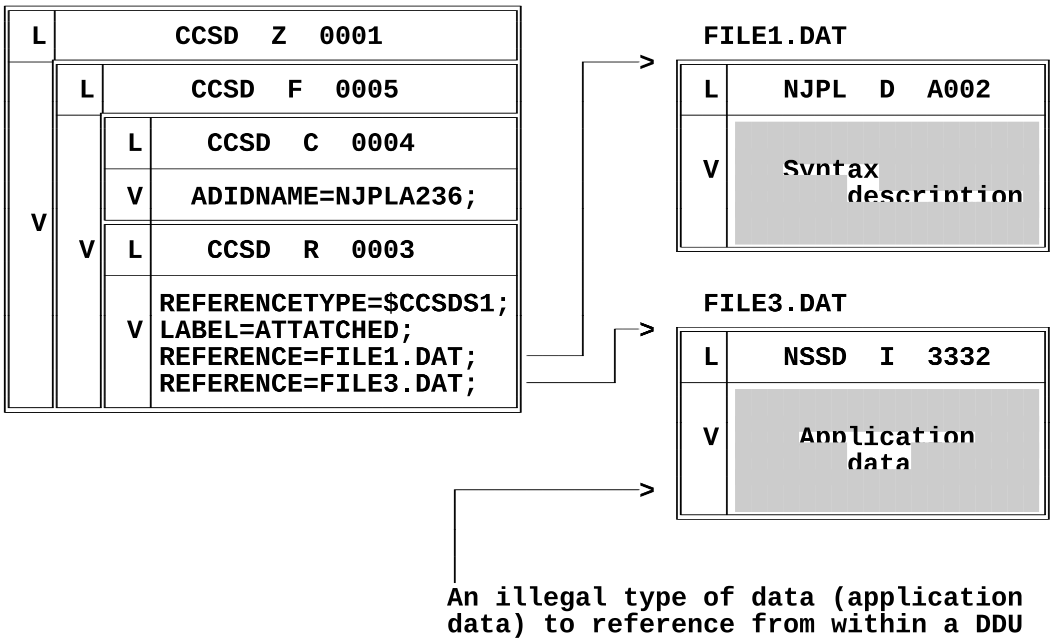


Figure 4-7: Example of an ERROR due to NOT Following the Structure Rules when Referencing an LVO

E. The delimitation of the Referencing LVO, and the LVOs within which it is contained, is not affected by the external data objects *(e.g., the octet count of a length delimited Referencing LVO does not take into account the octets of any external data object(s))*.

4.2.1 The REFERENCETYPE Statement

The **`REFERENCETYPE`** statement specifies the referencing environment by name. The convention of using a **`$`** as the first character of this name is intended to allow:

- The use of **`$CCSDS1`** - **`$CCSDSn`** to indicate that the referencing environment to be used is a proposed CCSDS developed referencing environment. Two such proposed referencing environments are currently defined.

- When such a proposed CCSDS defined referencing environment has been fully tested and evaluated, it will be incorporated into a CCSDS Recommendation. This means that a fully compliant CCSDS parser for a VALUE field with ADID = CCSD0003 must be able to understand the **`name_x`** values given in the **`REFERENCE`** statement, and access the necessary external data objects. Once the referencing environment is incorporated in a CCSDS

Recommendation then its name will not have a **$** at the beginning, i.e., it is an approved referencing environment.

4.2.1.1 Proposed Referencing Environment Specifications

The following two proposed CCSDS defined referencing environments have not been fully evaluated or tested at the time of issue of this Report. It is intended that these two proposed referencing environments be used. Assuming that they fulfil the needs of the user community, they will be approved in the form of a CCSDS Recommendation. The existence of only two proposed referencing environments at present should not be considered a restriction for the future. New reference environments can be introduced as user needs and technology evolve.

4.2.1.1.1 Basic Referencing Environment - `$CCSDS1`

`$CCSDS1` is the most basic referencing environment. It specifies a canonical filenaming convention that maps directly to all current major operating system filenaming specifications. This means that any SFDU products which use this referencing environment should be easily transferable across media with no truncation or clashes of filenames involved.

The specification is based on a simple "8 dot 3" filename, that is, a **`filename`** of maximum 8 characters with an optional **`extension`** of maximum 3 characters. These two segments of the filename may use the **`*`** and **`?`** wildcards to represent the remainder of their segment and a single character respectively. A directory path can also be specified. There is a maximum of 8 directory levels (separated by a `/` (forward slash)), each being a maximum of 8 characters long. If the directory path starts with `/` then it is relative to the highest level directory of the mounted volume or file system containing the reference LVO; if no leading `/` is present then it is relative to the location of the referencing LVO. The structure diagram in Figure 4-8 illustrates the described specification (for the complete specification of the **`$CCSDS1`** referencing environment see Reference [1], Annex F.1).

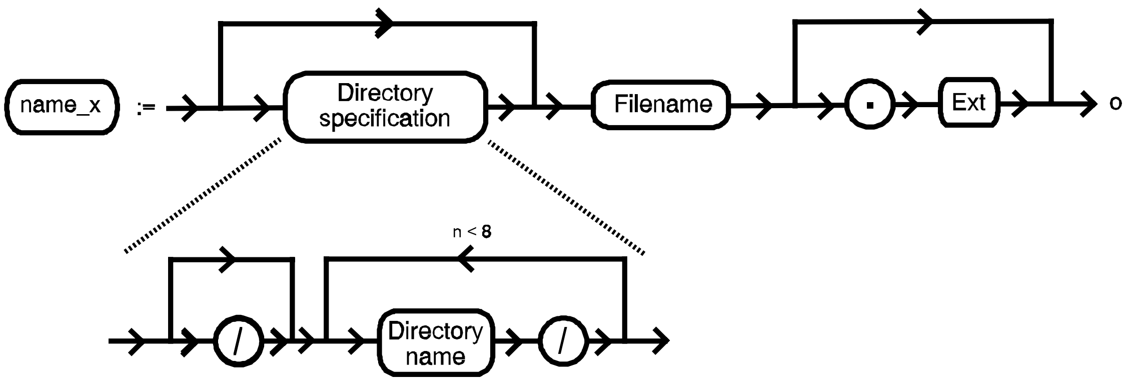


Figure 4-8: Structure Diagram of $CCSDS1 Name Specification

Table 3-5 shows a number of examples or how a **$CCSDS1** specified filename would translate to various popular filenaming conventions used by commercial operating systems:

$CCSDS1 filename	UNIX filename	MS-DOS filename	VAX VMS filename
FOOBAR.DAT	FOOBAR.DAT found in the same directory as the Referencing LVO	FOOBAR.DAT found in the same directory as the Referencing LVO	FOOBAR.DAT; found in the same directory as the Referencing LVO
FOOBAR	FOOBAR found in the same directory as the Referencing LVO	FOOBAR. found in the same directory as the Referencing LVO	FOOBAR.; found in the same directory as the Referencing LVO
FOOBAR.	An ILLEGAL **$CCSDS1** filename (if "." used then an extension must be present)		
MYDIR/FOOBAR.DAT	FOOBAR.DAT found in the MYDIR/ child directory of the directory where the Referencing LVO resides	FOOBAR.DAT found in the MYDIR\ child directory of the directory where the Referencing LVO resides	FOOBAR.DAT; found in the [.MYDIR] child directory of the directory where the Referencing LVO resides
\ADIR\FOOBAR.DAT	An ILLEGAL **$CCSDS1** filename (only forward slashes permitted)		
MYDIR/FOO.BAR.DAT	An ILLEGAL **$CCSDS1** filename (only 1 **.** (dot) permitted in a filename)		
/YOURDIR/MYDIR/FOOBAR.DAT	FOOBAR.DAT found in the /YOURDIR/MYDIR/ directory off the root directory of the volume	FOOBAR.DAT found in the \YOURDIR\MYDIR\ directory off the root directory of the volume	FOOBAR.DAT; found in the [YOURDIR.MYDIR] directory off the root directory of the volume
mydir/foobar.dat	An ILLEGAL **$CCSDS1** filename (only uppercase characters permitted)		
LONG_FOOBAR_NAME.DATA	An ILLEGAL **$CCSDS1** filename (only 8 characters permitted in the **filename** and 3 characters in the **extension**)		

Table 3-5: Examples of $CCSDS1 Referencing Environment Filenames

4.2.1.1.2 Extended Referencing Environment - $CCSDS2

This specification provides extensions to the $CCSDS1 specification for users who find the $CCSDS1 specification too restrictive. It maps directly to most major filenaming specifications, but is less portable than $CCSDS1. Essentially it is the same format as **$CCSDS1**, the difference being the number of characters that are permitted in each segment of the filename; each **directory name** is limited to 31 characters while the **filename** plus **extension** has a combined maximum of 30 characters (for the complete specification of the **$CCSDS2** referencing environment see Reference [1], Annex F.2). All the examples shown in Table 3-5 apply also for **$CCSDS2**, except that for **$CCSDS2** the last example is legal as the filename segments are within the permitted lengths.

Table 3-6 shows some examples of how the wildcarding features are applied (this is the same for both referencing environments). It shows a sample directory of files in the order that they are encountered by the operating system and returned to SFDU service software. On the right is the ordered list of filenames valid upon applying the reference environment filenames on the left.

Files in sample directory:	AAAA.DAT	ABXDE.SYS	FOO	ABCDE
	ABCDE.DAT	ABC	ABZDE.DAT	ABYYY.SAM
	FFGG	ABCDE.SYS	FOO.DAT	

Filename specified	Returned list of filenames			
*.DAT	AAAA.DAT	ABCDE.DAT	ABZDE.DAT	
*	FFGG	ABC	FOO	ABCDE
FOO.*	FOO	FOO.DAT		
AB*CD.DAT	Illegal as the **`*`** wildcard must be the last character in a **`filename`** or **`extension`** field			
AB*.SYS	ABXDE.SYS	ABCDE.SYS		
AB*/FOO.DAT	Illegal as the **`*`** wildcard is not permitted in a **`directory name`** segment			
AB*.S*	ABXDE.SYS	ABCDE.SYS	ABYYY.SAM	
AB?CD.*	ABCDE.DAT	ABXDE.SYS	ABCDE.SYS	ABZDE.DAT
AB?*	ABC	ABCDE		
AB*?.DAT	Illegal as the **`*`** wildcard must be the last character in a **`filename`** or **`extension`** field			
???	FOO	ABC		

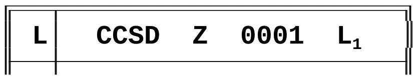
Table 3-6: Examples of Wildcards used in Referencing Environments

5 DELIMITATION TECHNIQUES

5.1 Delimitation By Length

In delimitation by length the length of the LVO is explicitly supplied in the LABEL of the LVO. This delimitation technique is intended for use when the size of the product is clearly known at the time of generation, and it is known that the transfer method will not add or remove any octets from the product. The technique also allows more accurate error checking than a number of other techniques, as if any one octet is accidently removed or added to the product then the nesting of the LABELs will clearly show this, i.e., a nested LVO will be completed either too soon or too late relative to a higher structure level.

Note: In the following diagrams a further parameter appears in the schematic of the LVO LABEL, namely the Delimitation Parameter, e.g.

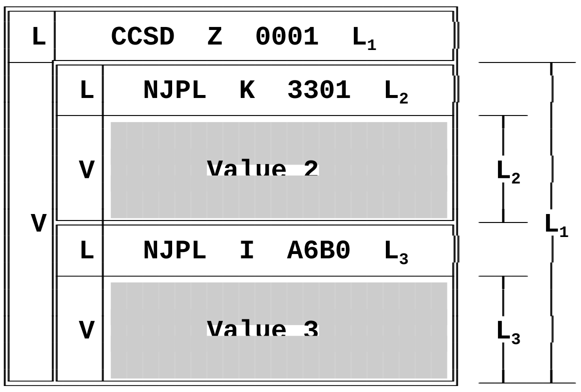


corresponds to an LVO LABEL with CAID=CCSD, Class ID=Z, DDID=0001 and a Delimitation Parameter=L_1. The Version ID in this example is assumed to be either "1", "2" or, "3" with a Delimitation ID = "A" or "B".

Delimitation by length may be used when:

- The total length of the unit or product is known;
- All the entities are physically stored sequentially on the same media.

It is illustrated in Figure 5-1. Two Simple LVOs (ADID = NJPL3301 and ADID = NJPLA6B0) are "enveloped" within an outer LVO. Note that the length sub-field of each LABEL encompasses its VALUE field. The VALUE fields of the inner LVOs are simply VALUE fields 2 and 3 while that of the outer LVO (VALUE field 1) is the combined lengths of the inner LVOs including their two LABELs. Delimitation by Length is well suited for building relatively small data products which can be buffered prior to output.



L_1=(L_2+20)+(L_3+20) in octets

Figure 5-1: Schematic of Length Delimitation

This technique is realised as follows:

- Using length, encoded in Restricted ASCII, i.e.:

  - Version 1, e.g. **CCSD1Z00000100001024**

  or

  - Version 3, Delimitation ID = A e.g. **CCSD3ZA0000100001024**

- Using length, encoded in binary, i.e.:

- Version 2	e.g. **`CCSD2Z000001nnnnnnnn`**
or	
- Version 3, Delimitation ID = B.	e.g. **`CCSD3ZB00001nnnnnnnn`**

Binary encoding permits lengths up to 2^{64}-1 octets, whereas RA encoding only permits lengths up to 10^{8}-1 octets. Obviously then, binary encoding is useful for large LVOs, whereas for any LVO shorter than 10^{8}-1 octets RA encoding is preferable for reasons of convenience and readability.

5.2 Other Delimitation Techniques

Other delimitation techniques are available which do not require the length of a product to be explicitly stated. These are useful for very large products or when the length of the product is not known at the time the first LABEL is generated (see Section 7.1.2 for further discussion).

These other delimitation techniques fall into two categories:

- Marker Delimitation, in which an explicit end marker (of a CCSDS defined format) is used to indicate the LVO end.

- End-of-File (EOF) Delimitation, in which the EOF services of the file system used are exploited.

These delimitation techniques are only available for Version ID = 3.

<u>Note</u>: *In the following schematic diagrams a further parameter is shown in the LABEL of the LVO, namely the Delimitation ID. Within Version 3 LABELs this is used to indicate how the Delimitation Parameter sub-field is used, e.g.*

L	**CCSD Z S 0001 ABABCDCD**

corresponds to an LVO header with CAID=CCSD, Class ID=Z, Delimitation ID=S, DDID=0001 and a Delimitation Parameter = ABABCDCD. The Version ID in this example is assumed to be "3".

Version 3 uses the Delimitation ID octet (octet 6) of the LABEL to denote the delimitation technique to be used. These "other" delimitation techniques are provided to cover the cases when the length of the data object is not known. The list of specific values of the Delimitation ID octet and their corresponding delimitation techniques is contained in Table 3-7 (overleaf). These techniques use, as necessary, the value contained within the Delimitation Parameter sub-field to complete the delimitation process.

Delimitation ID	Delimitation Technique
A	Length (specified in ASCII)
B	Length (specified in binary)
S	Marker Pattern
E	Sequential End-of-File
C	Contiguous End-of-File
F	Shared End-of-File

Table 3-7: Delimitation IDs

5.2.1 Delimitation by Marker Pattern - Delimitation ID = S

Delimitation by Marker Pattern employs a twenty octet marker pattern to indicate the end of the VALUE field. This delimitation technique is intended for use when the length of the VALUE field cannot be determined at the time of generation of the LABEL, or when transfer protocols or other processing may change the actual length of the LVO. Therefore an in-line 20 octet marker pattern is inserted when the VALUE field is complete. The first twelve octets of this marker pattern is a fixed pattern and must be '**CCSD$$MARKER**'; the remaining eight octets are user specified in the Delimitation Parameter sub-field of the LABEL. Thus, the marker pattern has the following format:

CCSD$$MARKERxxxxxxxx

The eight octet **xxxxxxxx** part of the marker pattern is the user specified part and can be any printable ASCII characters (decimal codes 32 to 126).

Delimitation by the marker pattern technique works in two ways, depending upon whether the LVO being delimited is a Simple or Compound LVO:

1. **Explicit Marker Pattern Searching (Simple LVOs)**

   With Simple LVOs (i.e., LVOs with Class ID = I, S, K, V, D, E, R or C), the end of the VALUE field is determined by carrying out an octet-by-octet search of the VALUE field until a twenty octet marker pattern, with the last eight octets the same pattern as that specified in the LABEL, is encountered. Figure 5-2 illustrates this case with an example of a class I LVO, where the user-specified part of the marker pattern is the ASCII string **ABABCDCD**.

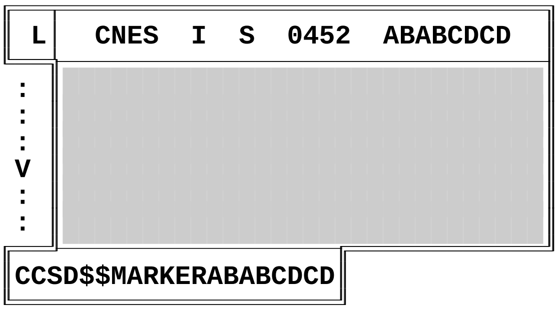


Figure 5-2: Example of Delimitation by Marker Pattern of a Simple LVO

2. **Navigation Through LABELs (Compound LVOs)**

   With Compound LVOs (i.e., LVOs with Class IDs = Z, F or U), successive LVOs contained within the VALUE field of the enveloping LVO are delimited and processed until a marker and not an LVO LABEL is encountered. Thus in this case the technique becomes a pairing or parenthesis matching technique. Although the user-specified pattern is not needed for pattern matching, as in case (1), it is useful for readability and error checking. Figure 5-3 shows two examples, the one on the left has a Compound LVO in which the Delimitation Parameter sub-field contains the name **WINDSCAT**, standing for Wind Scatterometer. The LVOs in this particular data object contain measurements from such an instrument, so the Delimitation Parameter sub-field performs a useful documentation function as well as allowing a cross-check that the LABEL and associated marker pattern have been correctly matched. The Simple LVOs in the VALUE field of the Compound LVO have Version IDs = 1 and 2, i.e., they are length delimited. The example on the right is a similar structure, except that one of the enclosed LVOs is delimited by marker pattern itself; this corresponds to case (1) in which a search through the VALUE field of the class I LVO is made for the marker pattern **CCSD$$MARKERALTIMINF**. This clearly shows the difference between the usage of marker pattern delimitation in Simple LVOs and Compound LVOs.

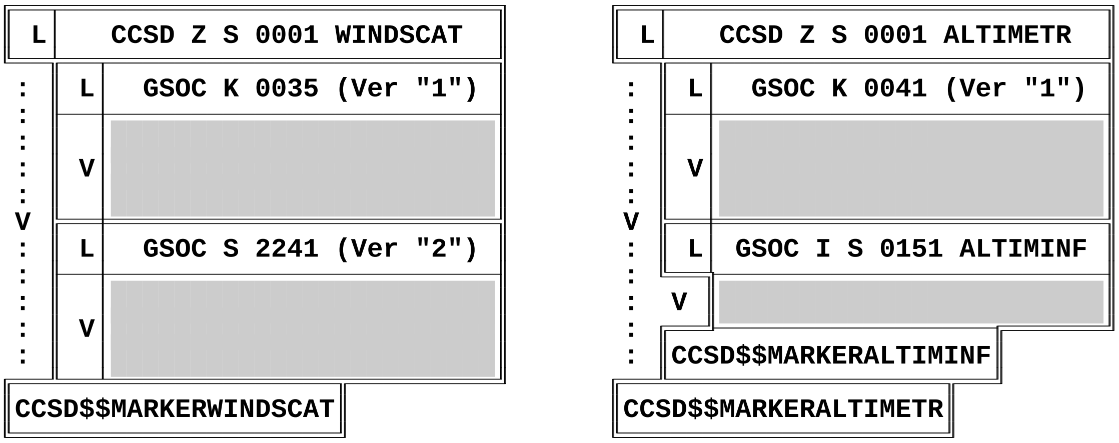


Figure 5-3: Examples of Delimiting Compound LVOs by Marker

Figure 5-4 (overleaf) shows a more complex example of marker pattern delimitation, building on the previous example. Here we have a product containing radar data from a spacecraft with a wind scatterometer and an altimeter. Data from each instrument is packaged in LVOs with Class ID = Z (EDUs) using marker pattern delimitation. The user-specified part of the marker patterns have the values **WINDSCAT** and **ALTIMETR**, incidentally as a side-effect providing a simple naming of each main unit in the product. The "outer loop" of the nesting uses the same technique, this time the name specified being **RADARSAT**. Obviously the technique can be extended to arbitrary numbers of **WINDSCAT** and **ALTIMETR** data units, a **RADARSAT** marker pattern being inserted as soon as the last unit has been put in place. This technique is particularly useful for multi-instrument spacecraft, in which operation time-lines for the various instruments differ and may change dynamically, according to, for example, day to day changes in mission plans or real-time changes in operations.

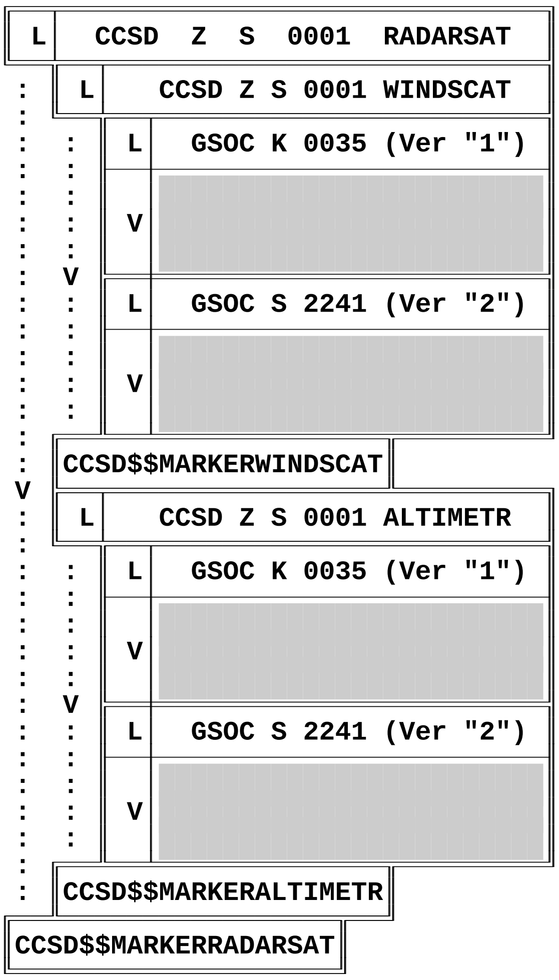


Figure 5-4: Example of Nesting of Marker Delimitation

5.2.2 Delimitation by End-of-File (EOF)

End-of File (EOF) is a technique common to many storage systems and is widely used for delimitation of data sets. However its physical implementation is specific to individual medium/operating systems.

Delimitation techniques that make use of EOF treat it conceptually without defining its implementation. It is assumed that the creation and recognition of EOFs are operations which can be performed for the specific medium/operating systems on which these LVOs are stored or transmitted.

Three different EOF delimitation techniques are available, covering the main common usages of EOFs on sequential and random access media.

5.2.2.1 Sequential End-of-File - Delimitation ID = E

In Sequential End-of-File delimitation an LVO is delimited by counting the number of EOFs encountered following the LABEL until the count reaches the number specified by the RA formatted integer value contained in the Delimiter Parameter sub-field of the LABEL. This applies straightforwardly for a Simple LVO as shown in Figure 5-5, where a Simple LVO consists of two sequential files, say on a magnetic tape.

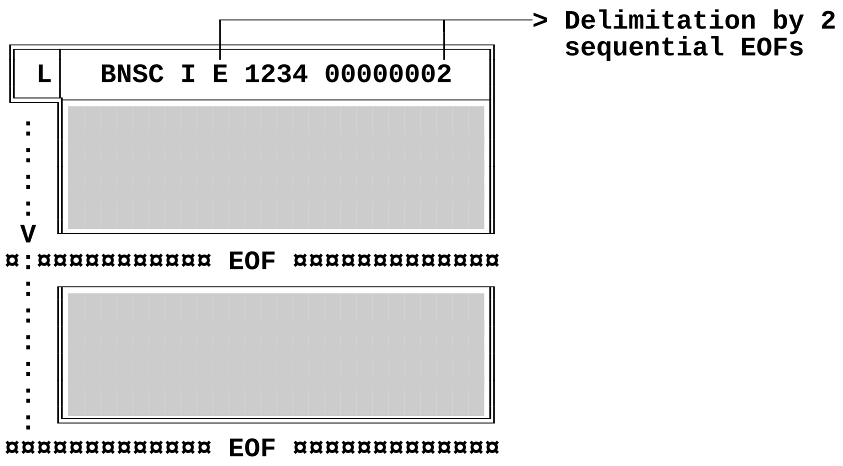


Figure 5-5: A single LVO Delimitation by 2 EOFs

For a Compound LVO, any EOFs encountered in the VALUE field of LVOs at lower structure levels or used to delimit LVOs at lower structure levels are ignored. Thus delimitation of the nested (or contained) LVOs must be completed first. Of course, if these lower structure level LVOs are themselves Compound LVOs also containing EOF delimited LVOs, the same procedure must be applied.

Figure 5-6 (overleaf) shows how nesting of Sequential EOF delimited LVOs within a Compound LVO works.

Delimitation by Sequential EOF is primarily intended for use with sequential access media, such as tape. If delimitation by EOF is used on random access media (i.e., disks) in which the concept of continuing beyond an EOF makes no sense, then only the RA character string **00000001** for the Delimitation Parameter is meaningful.

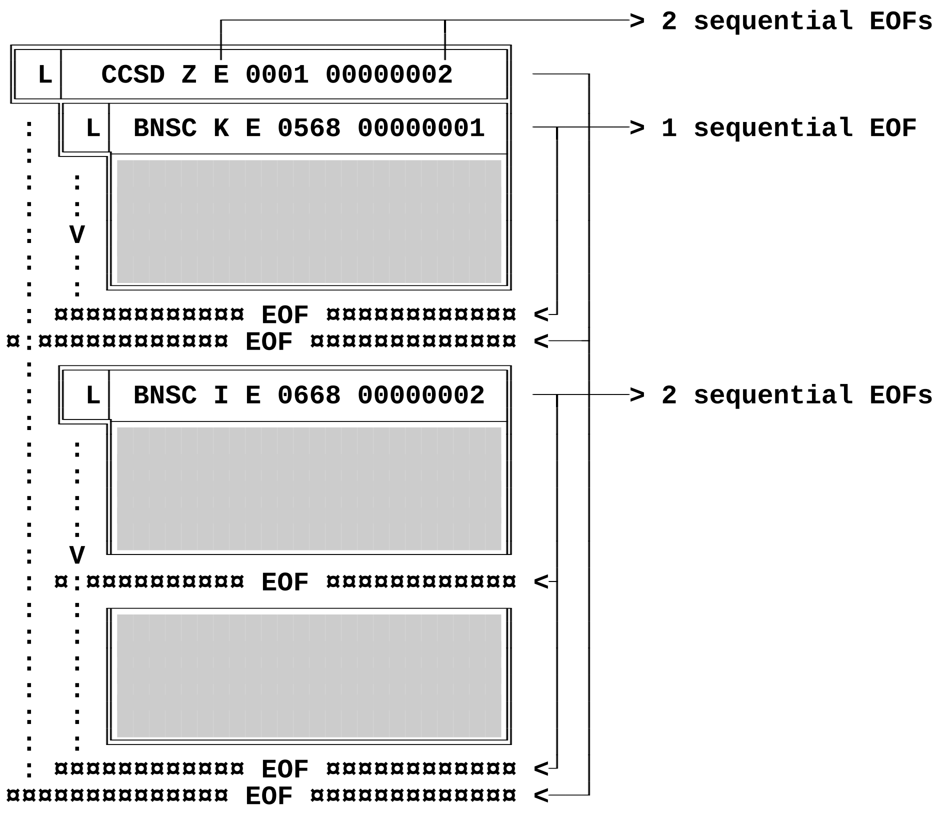


Figure 5-6: An Example of Nested EOFs

5.2.2.2 Contiguous End-of-Files - Delimitation ID = C

Delimitation by Contiguous End-of-Files is primarily intended for use with sequential access media, such as tape. An LVO is delimited by a contiguous number of EOFs. This number is specified by an RA formatted integer value contained in the Delimiter Parameter sub-field of the LABEL. This applies straightforwardly to a Simple LVO as shown in Figure 5-7, where a Simple LVO consisting of two files on a tape, is delimited by three contiguous EOFs. Note that in this example, the single EOF occurring between the two files is ignored.

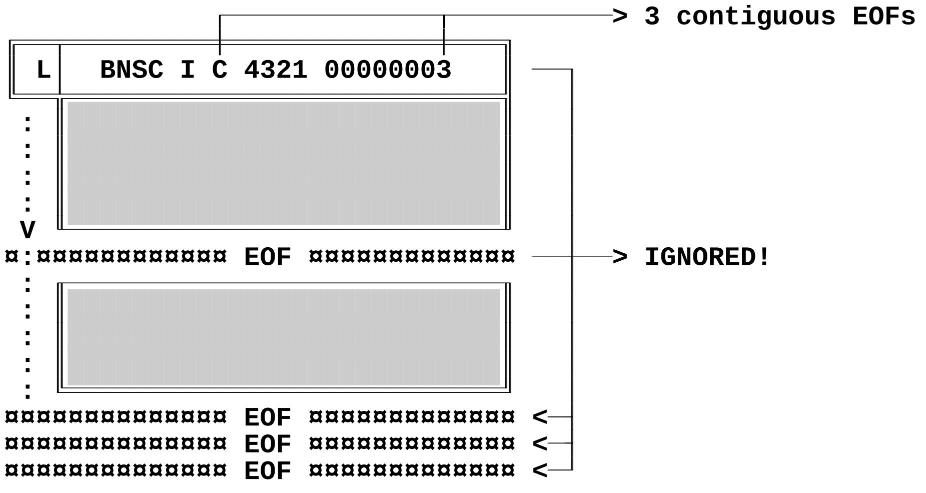


Figure 5-7: Delimitation by Contiguous EOFs of a Simple LVO

Similar rules apply when nesting Contiguous EOF delimited LVOs as for the Sequential EOF delimitation technique; that is, any EOFs encountered in the VALUE field of LVOs at lower structure levels or used to delimit LVOs at lower structure levels shall be ignored. Thus delimitation of the nested (or contained) LVOs must be completed first, as shown in Figure 5-8.

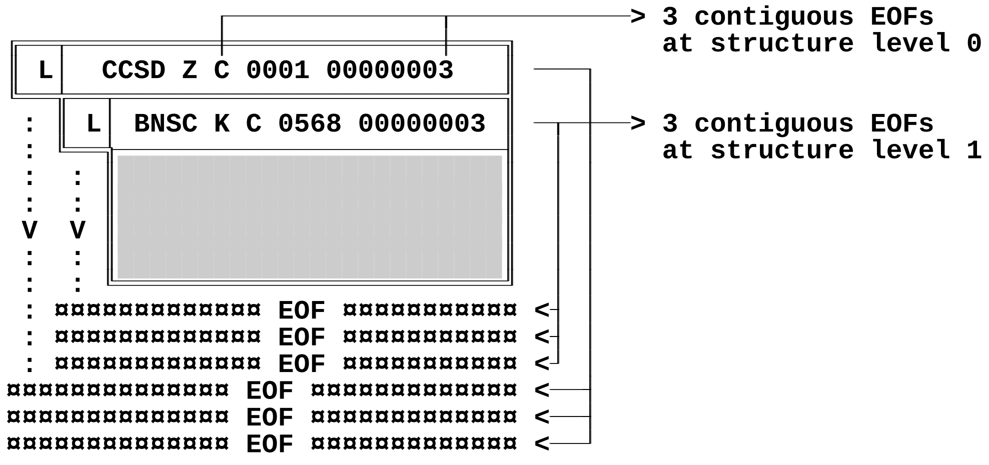

3 contiguous EOFs at structure level 0
L
CCSD Z C 0001 00000003
L
BNSC K C 0568 00000003
3 contiguous EOFs at structure level 1
V
V
EOF
EOF
EOF
EOF
EOF
EOF

Figure 5-8: An Example of Nesting Contiguous EOFs

For random access media (ie. disks) the same remark applies as for when delimiting by Sequential EOF, namely only a value of **`00000001`** for the Delimitation Parameter is meaningful. Since this type of EOF delimitation also does not permit sharing of EOFs, this LVO, when used on random access media, cannot be nested within any other LVOs.

5.2.2.3 Shared End-of-File - Delimitation ID = F

Delimitation by Shared End-of-File is primarily intended for use with random access media such as disks, in which the concept of continuing beyond an EOF makes no sense. In delimitation by Shared EOF an LVO is delimited by a single EOF. The EOF used in this type of delimitation remains available for use by higher structure levels if they use this type of delimitation. This may be applied multiply, that is, one EOF may be shared by any number of nested LVOs using this delimitation technique; they will all terminate at the same EOF. The value within the Delimitation Parameter sub-field of the LABEL must be set to the ASCII string **`00000001`**. This technique is illustrated by Figure 5-9 (overleaf).

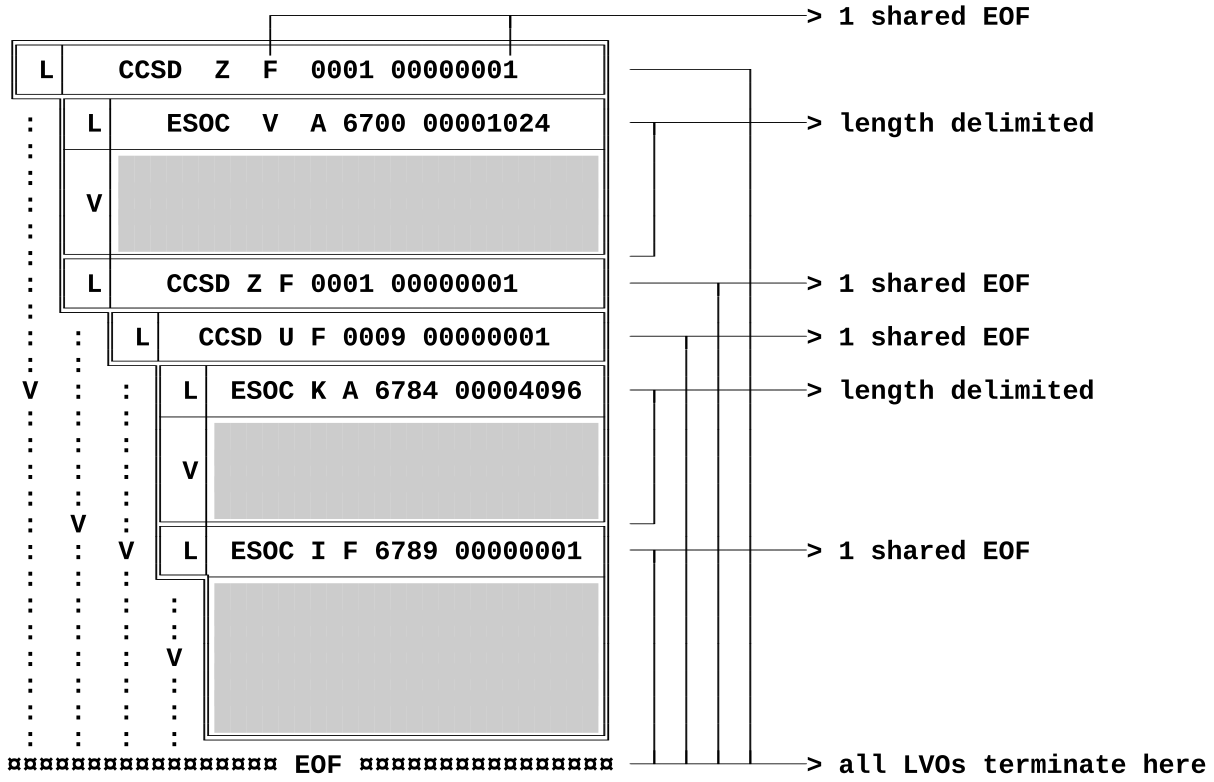


Figure 5-9: An Example of Use of Shared End-of-File Delimitation

5.3 Combining Delimitation Techniques

Delimitation techniques can be combined together. Only one rule is enforced when combining Delimitation techniques, and this is that it is not permitted to nest LVOs which are delimited by Sequential and Contiguous EOF within LVOs delimited by Shared EOF. Any other delimitation techniques can be combined in any way which works, i.e, which result in meaningful products which can be unambiguously interpreted.

For example, in an on-line data delivery system separate processes may produce a number of data objects to be delivered as one larger product. Each data object is produced as one LVO, i.e., the catalogue data (a class K LVO), the volume production data (a class V LVO) and the science data (a class I LVO). These separate processes know how much data they are producing and so delimit their generated LVOs by length (Version ID = 3, Delimitation ID = A). The overall data product building process does not know the size of LVOs that it will have to package, and so writes the EDU LABEL indicating a delimitation technique of Marker (Version ID = 3, Delimitation ID = S). After it has received the single LVOs and written them after the EDU LABEL, it writes the end marker pattern. This is an example of mixing Length and Marker delimitation techniques. The resulting SFDU product is shown in Figure 5-10 (overleaf).

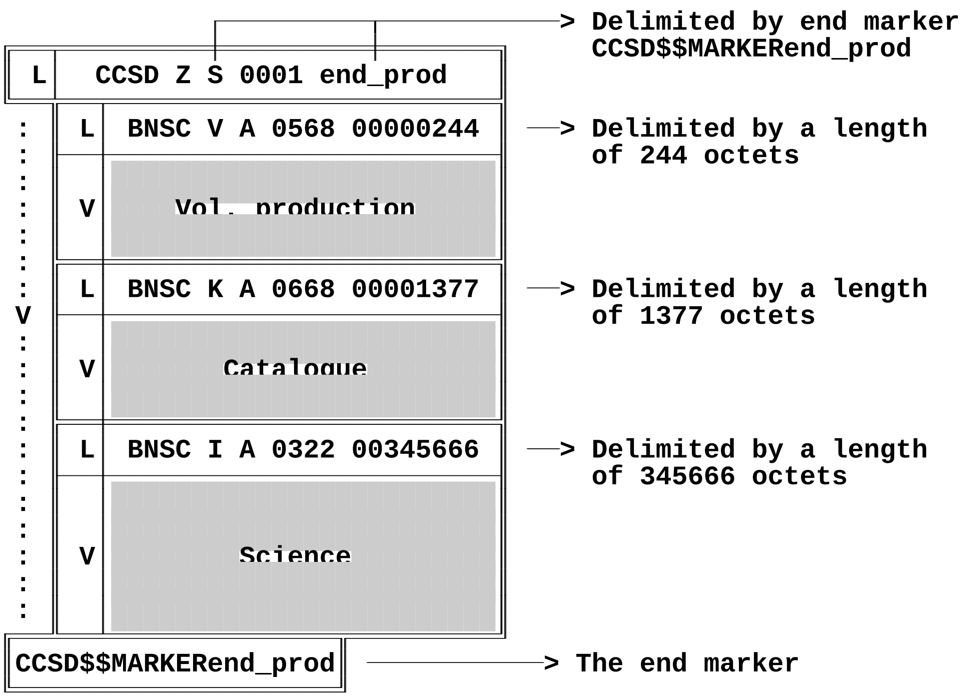


Figure 5-10: Mixing Delimitation by Length and Marker

An example of combining EOF type delimitation techniques is shown in Figure 5-11. This case could arise when the number of files which are on a (usually unlabelled) magnetic tape is unknown. Contiguous EOF delimitation is then used to indicate End-of-Volume and the individual files are delimited by Sequential End-of-File.

Quite complex combinations of delimitation techniques can be conceived.

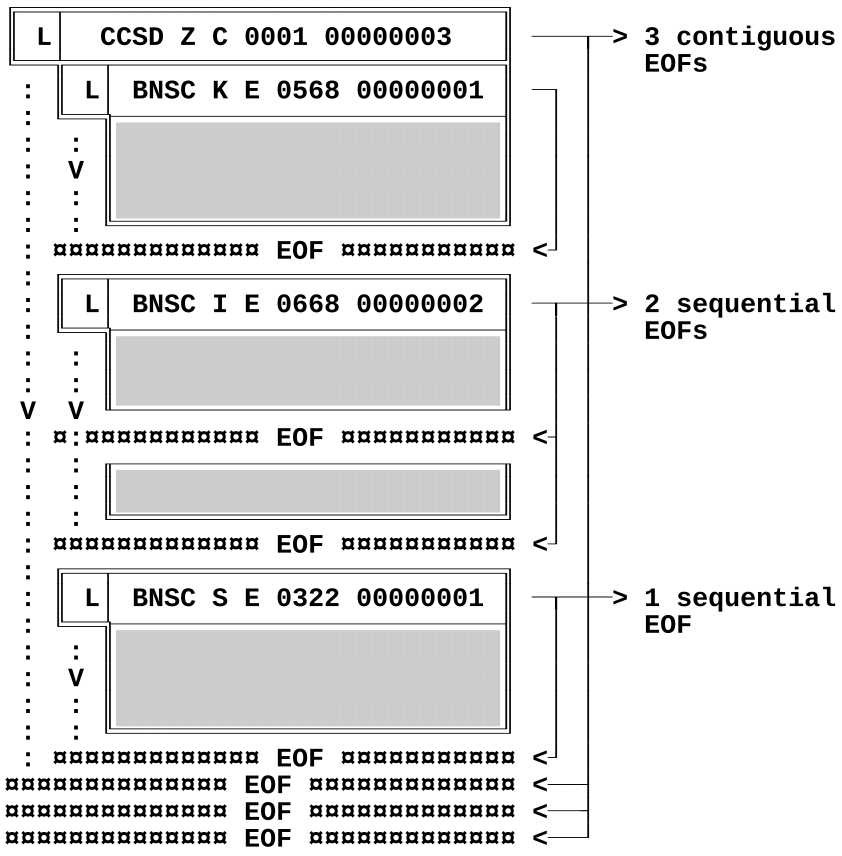


Figure 5-11: Mixing Delimitation by Contiguous EOF with Sequential EOF

Figure 5-12 shows an example using all three EOF delimitation techniques, as well as a length delimited LVO. All the LVO LABELs have Version ID = 3.

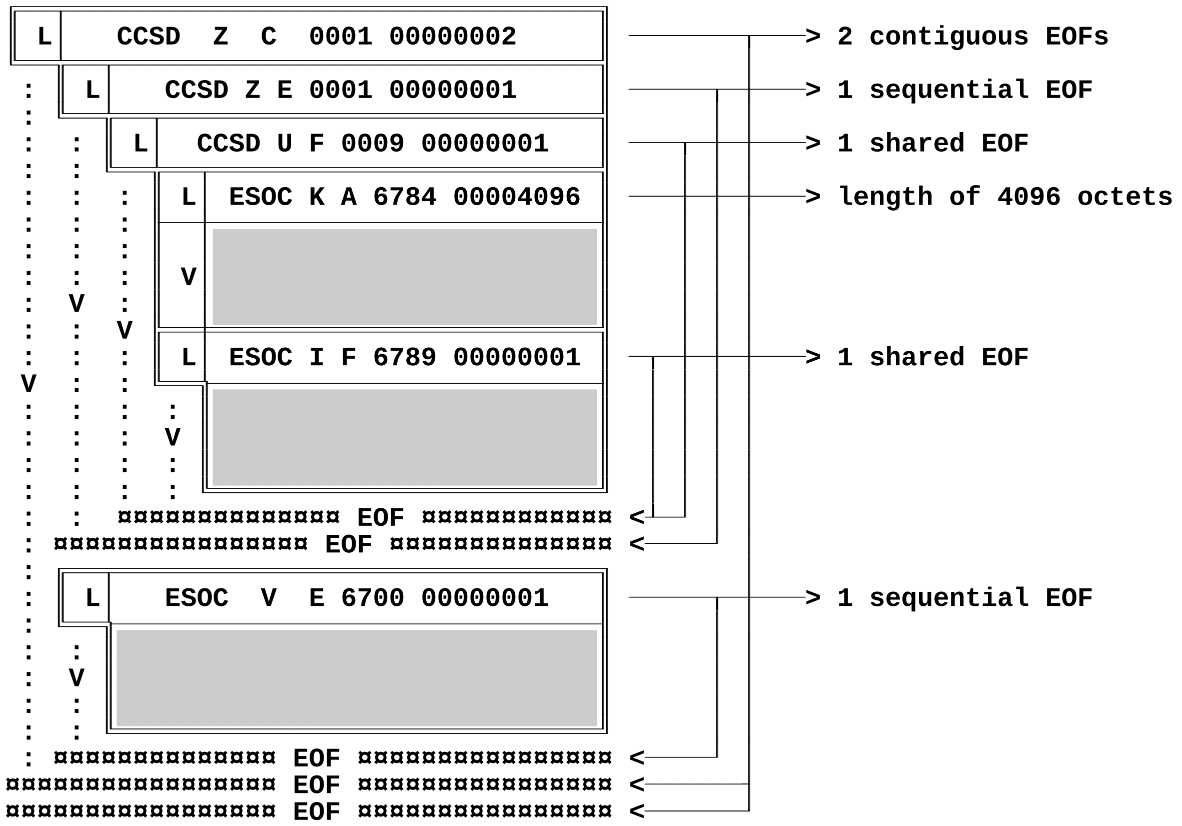


Figure 5-12: Mixed EOF Delimitation Techniques and Nesting

Figure 5-13 shows the overall structure in a schematic form, in which the EOF Delimitation IDs (E,C, or F) are followed by a number indicating the number of EOFs specified in the Delimitation Parameter sub-field. It is assumed that the data are read from left to right in the figure, the presence of an EOF in the input stream is indicated by an "**e**". The product has four structure levels numbered 0 to 3.

```
Structure |
  level   | >>>>>>>>>>>>>>> Octet stream >>>>>>>>>>>>>>>>
----------+---------------------------------------------|
    0     | C2-----------------------------------------|   C = Contiguous EOF
    1     |  E1-----------------------|E1-----------|  |   E = Sequential EOF
    2     |    F1---------------------||            |  |   F = Shared EOF
    3     |      A---------F1---------||            |  |   A = Length
----------+--------------------------------------------
                                     ||            |  |
                                     ee          eee <—— EOFs
```

Figure 5-13: Schematic of Mixed EOF Delimitation Techniques and Nesting

At the lowest structure level (level 3) there are two Simple LVOs, one delimited by length followed by one delimited by Shared EOF. These two are packaged within a Compound LVO also delimited by Shared EOF, therefore the first EOF in the octet stream delimits both these shared EOF delimited LVOs (see Reference [1], Section 3.3.4.6). The EOF does not delimit the LVO one structure level up as this uses Sequential EOF which ignores any EOFs encountered at lower structure levels (see Reference [1], Section 3.3.4.4). The next EOF in the input stream does delimit the first Sequential

EOF delimited LVO at level 1. The package so far complete is followed by another Sequential EOF delimited LVO, which again uses only 1 EOF. The EOFs used to delimit the Sequential EOF delimited LVOs do no count towards the delimitation of the highest level LVO, as Contiguous EOF delimitation also ignores any EOFs found at lower structure levels (see Reference [1], Section 3.3.4.5). Finally there are two contiguous EOFs to delimit the highest level (level 0) LVO and the package is complete.

If delimitation by length is combined with delimitation by EOF, as shown in Figure 5-14, then this is a legal structure. That is, the delimitation of the lower structure level LVO is deemed to be complete due to the end of the file, and then the higher structure level LVO is also completed. From an implementation point of view, this means that there must be a lookahead to notice that the end of the file has been reached (and the class I LVO has completed), and then following that, also the fact that the number of bytes specified in the class Z LVO for the length of its VALUE field has been reached.

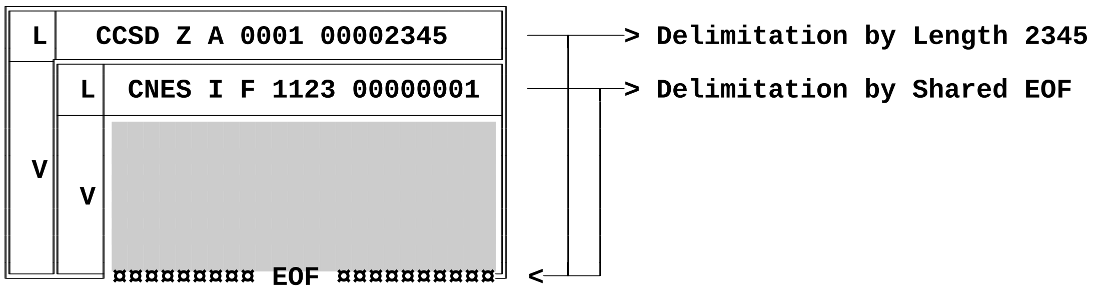
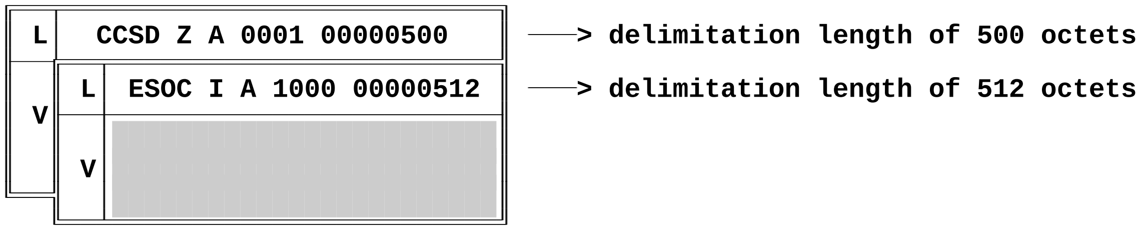


Figure 5-14: Nesting of EOF Delimitation within Length Delimitation

5.4 Potential Errors In Combining Delimitation Techniques

It is possible to devise combinations of delimitation techniques or delimitation parameters which while each LVO LABEL is syntactically correct, when the LVOs are combined they do not form a meaningful product and it is not possible to interpret the structure. For example:

- One of the most simple mistakes is when using only delimitation by length and not having the length of the VALUE fields add up correctly. For example in Figure 5-15, the class I Simple LVO has a VALUE field length of 512 octets, therefore the VALUE field of the class Z Compound LVO must be 512+20 (the size of the class I LABEL) = 532 octets, but the Delimitation Parameter value is only 500 and therefore obviously this is an error.



L
CCSD Z A 0001 00000500
V
L
ESOC I A 1000 00000512
V
> delimitation length of 500 octets
> delimitation length of 512 octets

Figure 5-15: Error in Length Delimitation Value

- A length delimited LVO containing an EOF delimited LVO for which the EOF appears before the end of the outer length delimited LVO has been reached. This is shown in Figure 5-16.

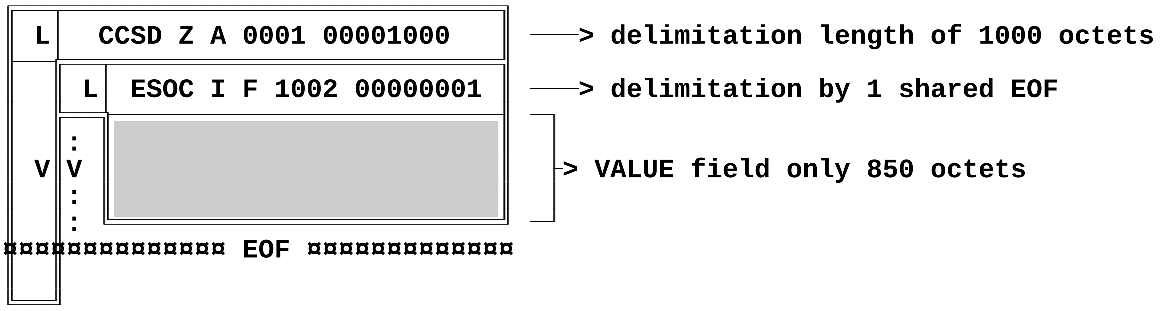

L
CCSD Z A 0001 00001000
→ delimitation length of 1000 octets
L
ESOC I F 1002 00000001
→ delimitation by 1 shared EOF
V
V
→ VALUE field only 850 octets
¤¤¤¤¤¤¤¤¤¤¤¤¤¤ EOF ¤¤¤¤¤¤¤¤¤¤¤¤¤

Figure 5-16: Error in Nesting EOF delimitation within Length Delimitation

This usage makes little sense in any case, since EOF delimitation is intended for the case when length is not known and therefore enveloping an EOF delimited object in an envelope of known length is not reasonable. The correct approach would be to use EOF delimitation for both inner and outer LVOs.

- A length delimited LVO is packaged within an LVO delimited by Shared EOF, as indicated in Figure 5-17. An EOF appears in the VALUE field of the length delimited LVO. While this is ignored by the length delimited LVO, the outer LVO recognises it and terminates prematurely. Therefore, this is an error.

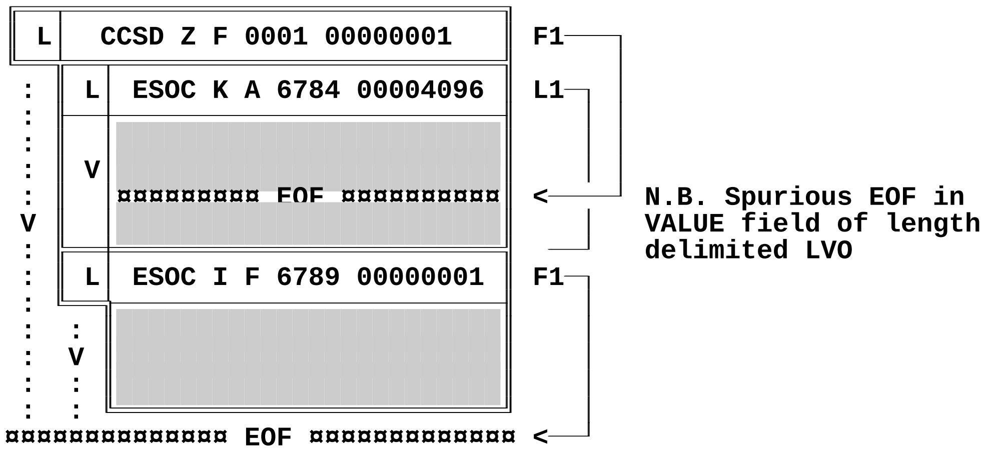


Figure 5-17: Erroneous Occurrence of an EOF within a Length Delimited LVO

Care should be taken to avoid such erroneous, doubtful or ambiguous cases.

6 THE COMPOUND LVOS

Note: In this section the term metadata is used to refer to data descriptions, i.e., "description metadata".

There are three forms of Compound LVO; the Exchange Data Unit (EDU), the Application Data Unit (ADU and the Description Data Unit (DDU). The functions of the EDU are summarised in Section 3.2.2.1. All SFDU products must be exchanged in an EDU with a Class ID = Z and an ADID = CCSD0001 in its LABEL. The other two Compound LVO are discussed in more detail in the following two sections.

6.1 The Application Data Unit

The purpose of an Application Data Unit (ADU) is to package application data instances together with any necessary ancillary data and identification (catalogue) information to represent a self-contained unit within the product for purposes of processing and analysis. Application Data Units only contain these categories of data and in particular do not contain metadata (i.e., data that would be stored in LVOs with Class IDs = F, D or E). Thus, in parsing a product in a setup where DDUs are supplied with the data, the user can always be sure that the ADUs will only contain application data and no metadata. Of course, the same packaging function can be realised using an LVO of Class ID = Z as the envelope, but in this case a process, looking say for metadata only, would have to "open" all the nested SFDUs down to all nesting levels in order to ensure that no metadata is missed. On the other hand it could skip over ADUs knowing that there is no metadata within.

An example of an ADU is shown in Figure 6-1. This sample structure is a class U LVO (an ADU) within an outer EDU package (LVO with ADID = CCSD0001 and Class ID = Z). Contained within the VALUE field of the ADU are three LVOs containing an instance of application data, some supplementary data, and the pertinent identification or catalogue data.

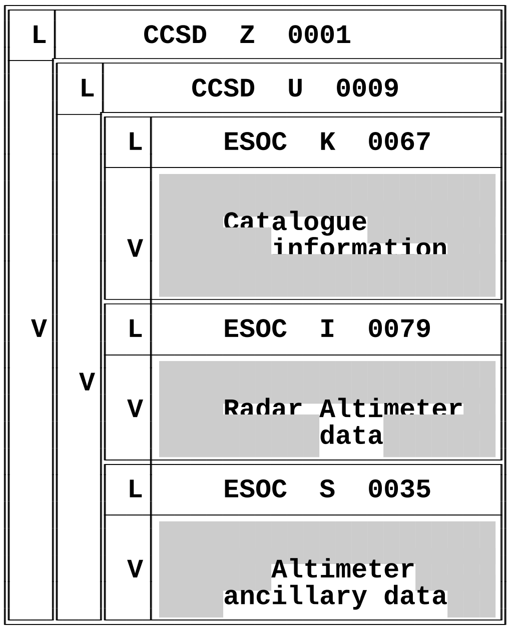


Figure 6-1: Example of an ADU

ADUs may also be nested as shown in Figure 6-2, and must always be carried within an EDU (a class Z LVO) product.

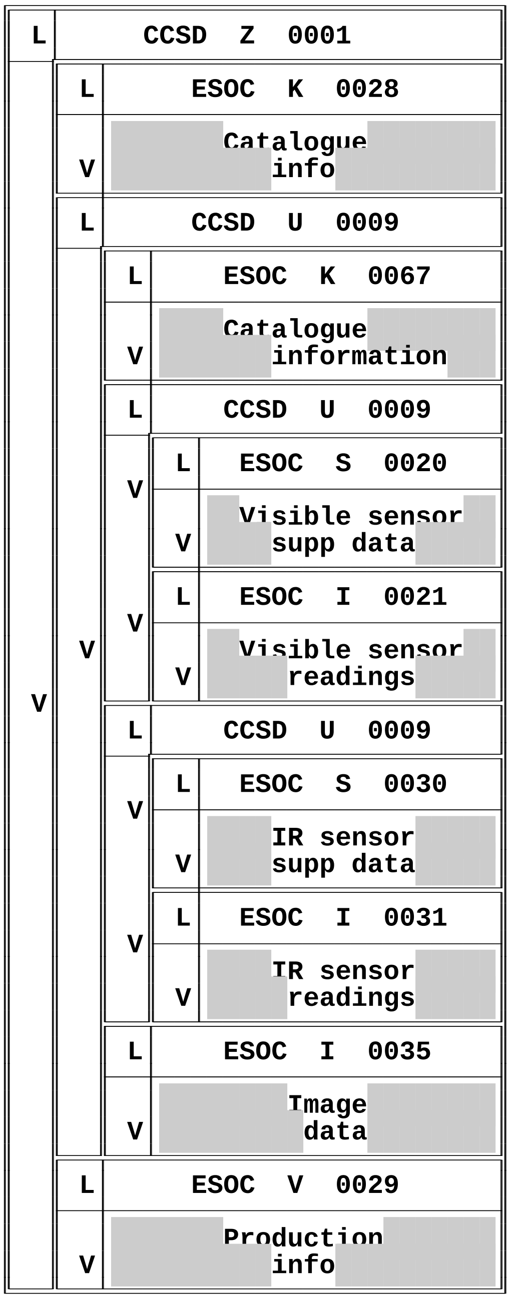


Figure 6-2: Example of Nested ADUs

In the current Recommendation (Reference [1]) the ADID = CCSD0009 is provided for building ADUs. Since an ADU is intended for carrying information data instances, only LVOs with Class IDs = U, I, S and K are permitted within an ADU's VALUE field. The service classes R and C are also permitted, although CCSD0009 does not define a use for LVOs with Class ID = C. There may be further ADIDs defined in the future which utilise it.

6.2 The Description Data Unit

6.2.1 Overview

Description Data Units (DDUs) are used for packaging together metadata. The DDU also provides a mechanism for linking ADIDs and the corresponding data descriptions as illustrated in Figure 6-3. The left hand side of the diagram shows a simple EDU containing an LVO of Class ID = I. The links to a schematic data base are shown; this data base notionally contains CCSDS defined ADIDs and registered descriptions (e.g., ESSD1233). In practice the CCSDS-defined ADIDs will be incorporated in service software of some type, so the pointing is to that software. On the other hand, the registered descriptions will be stored in some kind of physical database on the system processing the SFDU. The ADID of any LVO should give access to a complete description of that LVO's VALUE field. This might be either a set of structure and construction rules defined in Reference [1] or a description supplied by the producer of the data. The problem is that the data description does not itself contain the ADID of the data to which it refers. Its ADID will describe how to interpret its own VALUE field. To solve this problem a DDU carries the ADID of the described data along with the data descriptions.

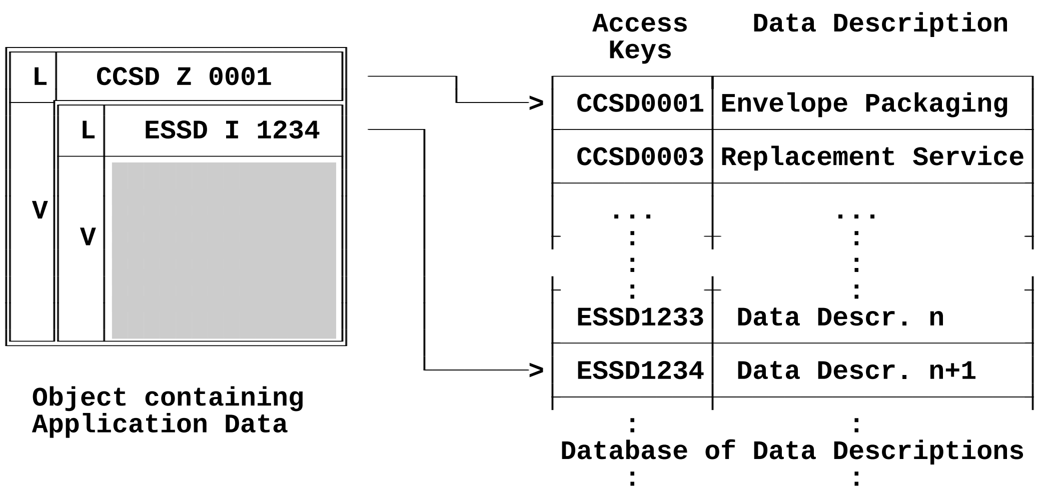


Figure 6-3: Referencing of Data Descriptions

The essential contents of a DDU are:

1. A mandatory class C LVO containing the ADID of the data unit to which that DDU applies. This is used for data description identification. This LVO may also optionally contain references to other metadata not carried within the DDU

2. Data Description LVOs (metadata), that is class D, E or S LVOs.

An example of a DDU is shown in Figure 6-4.

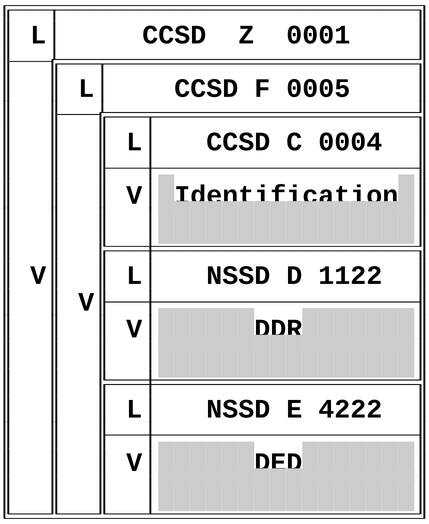


Figure 6-4: Example of a DDU

Figure 6-5 gives an example of how a DDU relates to the data it describes. In Figure 6-5, the second and third LVOs (Class IDs = D and E) form the data description itself, whereas the first LVO (Class ID = C) contains the ADID under which this data description is registered. The class I LVOs on the left hand side of the diagram use this ADID to reference the class D LVO (Data Description Record) and the class E LVO (Data Element Dictionary) on the right hand side of the diagram. The VALUE field of the class D LVO is written in FORTRAN, and its ADID (NJPL L006) will reference a description of this language. The class E LVO is written in a form of structured English, which will be defined in the description registered under NSSD4222.

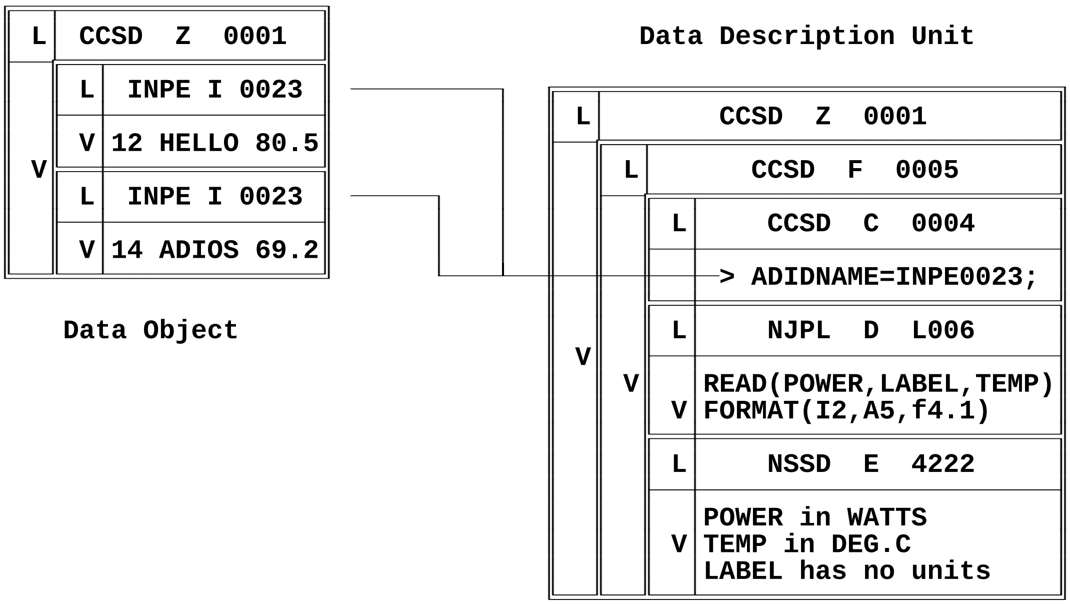


Figure 6-5: Data Object/DDU Relationship

The data description information should normally be in place on the user's system available for interpretation of the application data. An example of an automated implementation is as follows:

> The data description information upon arrival is logged into a library or database system, keyed on the ADID extracted from the class C LVO. When a data object with that ADID is received, the corresponding data description information is in principle retrieved from the library, using this ADID as a key. If the description is composed of LVOs whose labels contain CCCDS ADIDs, then these LVOs can be parsed using generic SFDU software, since they conform to CCSDS standards. However LVOs whose labels contain non-CCSDS ADIDs can only be directly parsed if suitable (normally local) software has been prepared to parse according to this description. Thus for maximum automation in parsing and interpretation in open systems, it is recommended that the description LVOs be expressed using CCSDS standard languages and formats, where this is possible.

DDUs are built using the construction rules associated with ADID = CCSD0005.

6.2.2 Data Description Identification

A CCSDS defined ADID is provided for describing the VALUE field of the first LVO within a DDU. This is ADID = CCSD0004. The LABEL field thus has ADID = CCSD0004 and Class ID = C. ADID = CCSD0004 indicates that the VALUE field will consist of one or more "parameter=value" statements. Four parameter names are permitted, as follows:

`ADIDNAME = value`	(mandatory)
`DEDID = value`	(optional)
`DDRID = value`	(optional)
`SUPID = value`	(optional)

Note: Reference [1] should be consulted for exact syntax and rules.

The statement **`ADIDNAME=value`** supplies the Authority and Description Identifier (ADID) under which the DDU is registered.

The statement **`DDRID=value`** supplies the ADID under which a Data Description Record (DDR) to be used has been stored. Several such statements may appear in this VALUE field, this situation corresponds to the use of alternative Data Description Languages (DDLs) to describe the same data.

The statement **`DEDID=value`** supplies the ADID under which a Data Entity Dictionary (DED) to be used has been stored. The order of precedence of these referenced DEDs is the order of occurrence in the DDU. Reference DEDs have lower precedence than DEDs actually included within the DDU.

The statement **`SUPID=value`** supplies the ADID under which Supplementary Metadata Data (in a class S LVO) to be used has been stored. Several such statements may appear in this VALUE field.

The **`value`** provided in each statement is an eight RA character text string, specifying an ADID (CAID (four bytes) followed by a DDID (four bytes)).

Figure 6-6 illustrates referencing of a DDR and a DED. In this example, the DDR with ADID = NJPLL010 and the DED with ADID = NJPLL009 have been registered separately. The LABEL of the class I LVO on the left hand side of the diagram indicates that the DDU describing it will be identified by ADIDNAME = NJPLA236. Inspecting the DDU itself, on the right hand side of the diagram it can be seen that the required DDR(s) and DED(s) are not carried in the DDU, but referenced to in other DDUs, therefore these DDUs must then be examined. Any DDRs in the DDU with ADID = NJPLL010 are logically included and also any DEDs in the DDU with ADID = NJPLL009. Of course when the DDUs with ADIDs = NJPLL010 and NJPLl009 are examined they may also refer to other DDUs and hence the chain would continue until actual class E or D LVOs are found.

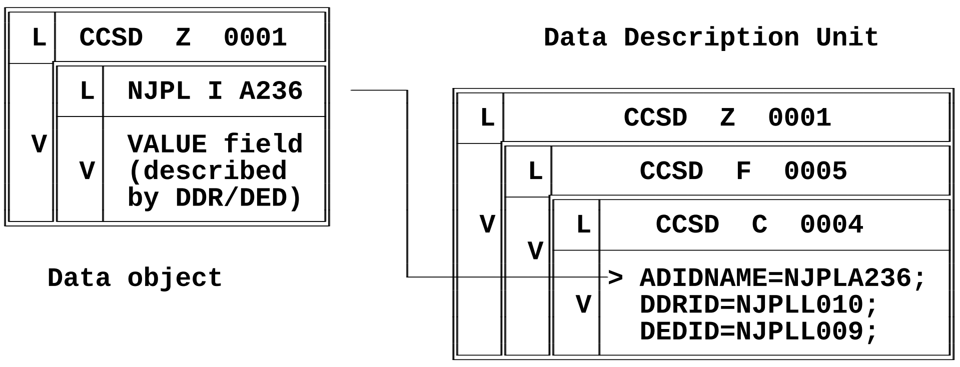


Figure 6-6: Example of DDU with Referenced DDR/DED

6.2.3 Data Description Records - LVOs with Class ID = D

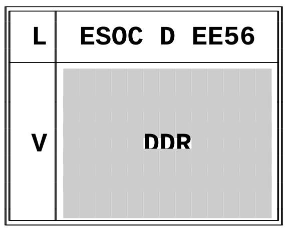


Figure 6-7: DDR LVO Example

The Data Description Record (DDR) provides the information which describes the syntax of a data object, and provide the means by which a data object can be formulated or parsed. This set of statements identifies the data entities and describes the format (data representation) that is used to express its value in the data object. In Figure 6-7, the ADID ESOCEE56 indicates the language in which the DDR is expressed. A DDR may be composed in any Data Description Language (DDL), which is a language used to describe data objects. It is desirable to use a machine interpretable DDL to support the automated parsing and construction of data products.

6.2.4 Data Entity Dictionaries - LVOs with Class ID = E

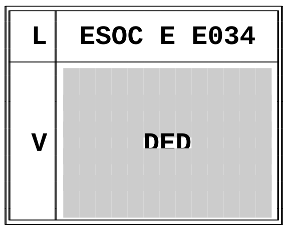


Figure 6-8: DED LVO Example

A Data Entity Dictionary is the repository of the definitions of the vocabulary used in an information system. A DED enables users separated by discipline, geography or time to share the same semantic view of that data. The primary requirement of a DED is to provide complete definitions of the vocabulary used in the data product labels, in the metadata for the data product, and in the descriptive detail of data objects. Each entry in a DED will typically define a data entity, give information such as name, unit, type (e.g., digital, analog, etc.), domain, etc.

6.2.5 Supplementary Metadata - LVOs with Class ID = S

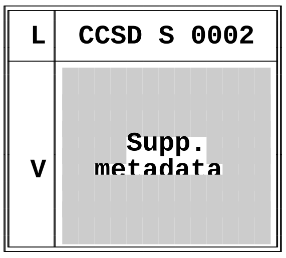


Figure 6-9: Supp. Metadata Example

When a class S LVO is used within a DDU, the supplementary data is said to be instance independent metadata, that is data which is supplementary to data descriptions and not supplementary to application data, as is the case when class S LVOs appear elsewhere in an SFDU structure. Supplementary metadata will typically be used to provide additional semantic material that is not appropriate for a DED. However the concept of supplementary metadata also includes background documentation such as instrument descriptions. Figure 6-9 shows a class S LVO which contains supplementary metadata that is expressed as English text, as indicated by the ADID = CCSD0002.

7 REQUIREMENTS, RATIONALES AND COMPLIANCE

The requirements which must be satisfied by the SFDU structure and construction rules are stated in Section 7.1. Each requirement is numbered to permit cross-checking with the implementation technique provided to satisfy it.

A Compliance Table showing how the SFDU Structure and Construction Rules (Reference [1]) meets the requirements appears in Section 7.2.

7.1 Requirements

7.1.1 Product Building or Data Packaging

1. Provide the means to generally classify or categorise data units.

Rationale:

Certain broad categories of data are common in data products:

(i) The application data itself;

(ii) Data supplementary to the application data such as instrument housekeeping, pointing data, platform housekeeping;

(iii) Catalogue or identification data (giving, for example, dates/times of measurements, orbit number, overall data quality, etc.);

(iv) Production information (e.g., date of product generation, producer system, volume labelling information).

Recognition of these categories can help the systematic design of products. Processing is also assisted, since processing systems can be designed to deal with the various categories of data, which can be recognised and routed to the relevant modules.

2. Provide the means to structure data objects within a product into a hierarchy (i.e., to allow nesting of the product's component units).

Rationale:

Data products are frequently hierarchical because:

(i) Application data may be packaged at several levels, eg. major-frames and minor-frames, images and image lines. Figure 7-1 (overleaf) shows this case schematically;

(ii) Products may contain output from several instruments each with a hierarchical structure mentioned in (i) above. Figure 7-2 (overleaf) shows an example of this case;

(iii) Data streams are frequently merged for transmission purposes and then split out again on reception.

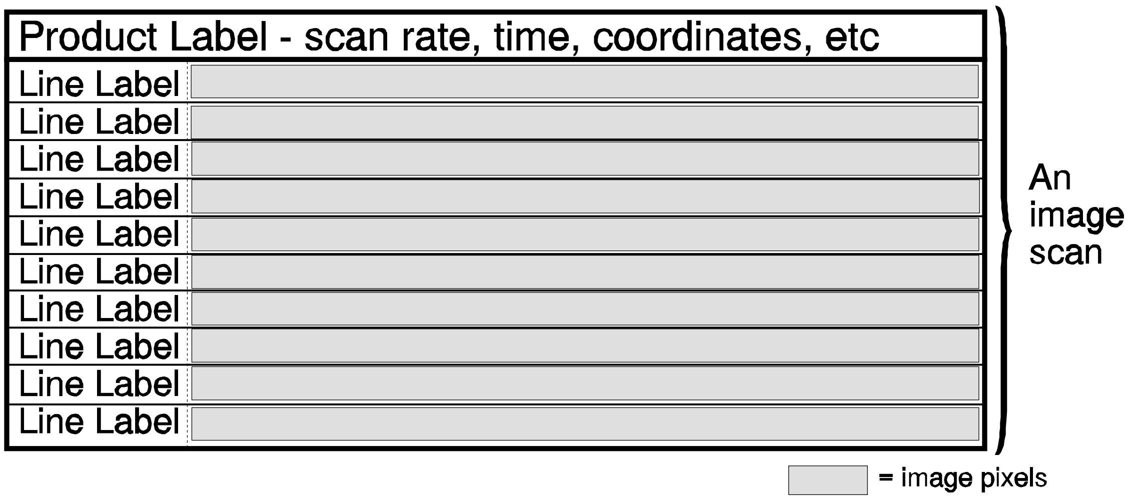


Figure 7-1: Schematic of Visible Image Scan

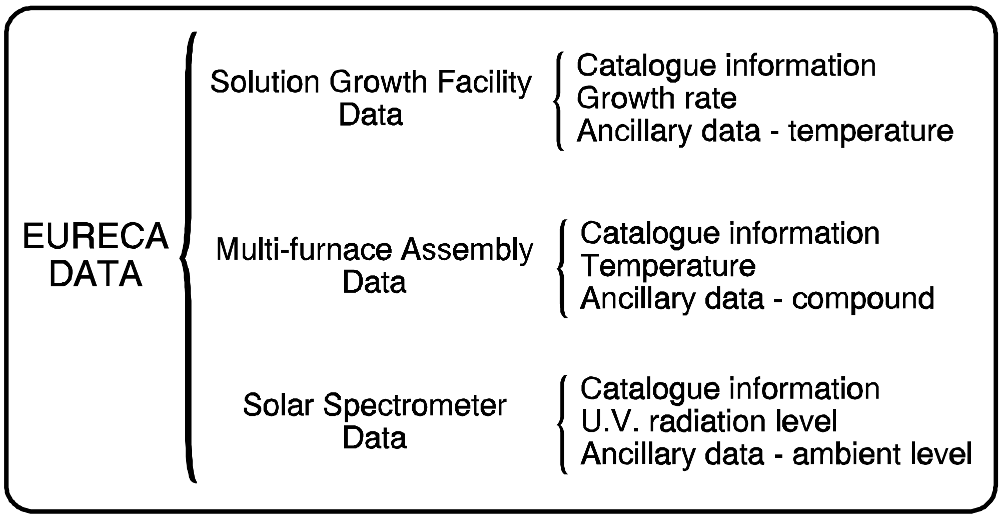

EURECA DATA
Solution Growth Facility Data
Catalogue information
Growth rate
Ancillary data - temperature
Multi-furnace Assembly Data
Catalogue information
Temperature
Ancillary data - compound
Solar Spectrometer Data
Catalogue information
U.V. radiation level
Ancillary data - ambient level

Figure 7-2: Schematic of Eureca Product Structure

These two example cases may be combined thus creating quite complex hierarchies.

3. Allow packaging of data on common media:

   3.1 Provide means to package contiguous data units into a data product.

   Rationale:

   This is the standard way of building data products, where files, records, blocks or other components are put together contiguously on a magnetic tape, disk or other physical support. Alternatively the product may be transmitted over data lines as a continuous stream.

3.2. Provide means to reference data units which are not contiguous with the rest of the product.

Rationale:

(i) *For a large data product it may not be convenient to put the data all together contiguously.*

(ii) *Several products may need to reference some common data.*

4. Provide means to include existing data objects unchanged into a product complying with the Recommendation.

Rationale:

Existing data forms for which there is a large base of installed processing/analysis software must be supported.

7.1.2 Data Delimitation

5. Allow indication of data unit length by an explicit indication.

Rationale:

(i) *Data objects of fixed length, in many cases containing an embedded length indication, are the commonest type of object in both real-time and off-line systems (e.g., most communications protocols break the transported data up into fixed length blocks).*

(ii) *It is traditional for data products to contain length indications in standard headers.*

(iii) *Length provides a good means of steering from one unit to the next without the need for examining data contents, marker searches etc.*

6. Provide means independent of length to delimit data units or entire products when the length of the product is not known in advance.

Rationale:

(i) *Science data product generators collect science and supplementary products from a number of sources. Often these collections are too large to be buffered within a machine so that total size cannot be determined in a single pass. This can be handled by breaking the data produced into fixed length blocks or by generating a marker when the product is complete.*

(ii) *Some products are created by a human using a word processor or text editor. The related software typically does not have facilities to count the total number of characters and control bytes of the resulting data object. Performing the counting manually would be time consuming and error prone.*

7.1.3 Data Description

7. Provide the means to separate instance-independent metadata from the data it describes.

   Rationale:

   This means that metadata can be transmitted once only and then used many times. Otherwise retransmission of fixed metadata would be inefficient, particularly with complex products.

8. Allow this metadata to be separated into:

   8.1 Syntax metadata (i.e., that giving order, location, organisation of the data being described).

   8.2 Semantic metadata (i.e., meaning) of named data elements.

   Rationale:

   (i) It is now common to distinguish between data syntax (format, representation, etc.) and semantics (data element dictionaries, etc.).

   (ii) Several communities may use a given set of data but wish to use different names and terminology. These differences can be reflected in different semantic metadata associated with the same syntactic metadata. That is each community uses the "native" semantics of each format and thus requires the "dictionary" for each format.

   (iii) A given community may wish to establish a common set of semantics for use with different formats. This is the opposite approach than taken in (ii).

9. Provide a means to permit association of data instances with their corresponding metadata.

   Rationale:

   There must be a means of making the correspondence between a data instance and its (in principle separate) data description.

10. Provide a means to describe the overall structure of a product.

    Rationale:

    A Product may be more than the sum of its parts. Conventionally products have been described in Data Interchange Documents and Interface Control Documents which formally describe a complete product or set of products.

7.2 Requirements Implementation Compliance

The requirements described in Section 7.1 have been met by a set of SFDU techniques which have been outlined in Sections 3 to 6.

The Compliance Table (Table 7-1) shows how the requirements are satisfied. In particular, column 3 of this Table gives the object or construct by means of which that requirement is met. The means may consist of a particular set of construction rules. Column 4 gives the reference to the subsection where the implementation(s) of that requirement is discussed.

No	Requirement	Satisfied by	Section
1	Classify or categorise data units	Class identifier in LABEL	3.4.1
2	Structure data objects within a product into a hierarchy	Packaging techniques	4
3	Package contiguous data units into a data product	Envelope packaging	4.1
4	Reference data units which are not contiguous	Replacement services	4.2
5	Explicit indication of data unit length	Delimitation techniques specified in Version 1, 2 and 3 with Delimitation ID = A or B	5.1
6	Delimit data units or products when the length of the product is not known	Version 3 delimitation techniques not included in requirement 5 above	5.2
7	Separate instance-independent metadata from the data instances it describes	Description data unit	6.2
8	Allow metadata to be split into: syntax metadata semantic metadata	Description data units contents: DDR LVO (Class ID = D) DED LVO (Class ID = E)	 6.2.2 6.2.3
9	a) Permit association of data instances with their corresponding metadata b) Make descriptions self identifying	The ADID concept Class C LVO (within a DDU) carrying ADID of described data	7.1 6.2.1.1
10	Provide a means to describe the overall structure of a product	Not formally satisfied. It can be informally satisfied by including in the product a Class = S LVO containing a description of the product structure in e.g. English or graphic form.	-

Table 7-1: Requirements Compliance

8 EXAMPLE SFDU APPLICATIONS

Given a complete outline of the rationale and rules for SFDU structures, the reader is now in a position to understand some examples drawn from real projects. This shows how the concept can be used in practice. The examples are discussed in sufficient detail for the reader to understand how the techniques have been applied. In general, full details of data structures down to the byte level are not presented - for this the reader will have to consult the relevant project documentation.

The examples given are:

- European Retrievable Carrier (EURECA) Data Disposition System, which is an example of a fairly simple SFDU envelope packaging system, using length delimitation. This is also a case of an on-line or interactive SFDU application. Briefly discussed in this context are similar systems being developed both by NASA and ESA for the International Solar Terrestrial Programme(ISTP).

- International Halley Watch, which shows the extensive use of the Replacement Service (file referencing) for archive data sets on CD-ROMS.

- Data Description Package Registration and Dissemination, apart from giving a good scenario for registration of data descriptions, this example shows extensive usage of marker delimitation for data sets prepared on a word processor.

- Data Description Language Usage for Data Description Record Writing. This shows an example of using the data structure declaration parts of the Ada programming language to describe data logically and also physically for any particular medium/operating system.

  (The examples represent the status of the projects concerned at the time this document was written)

These examples demonstrate all the basic aims of the SFDU concept and are drawn from real projects. In particular they show, in collaboration, the capability of complete end-to-end automated generation and interpretation of SFDU products, in both on-line and off-line applications.

8.1 European Retrievable Carrier (EURECA)

The European Retrievable Carrier (EURECA) is due to be launched by the Space Shuttle Atlantis in 1992. It will fly in a low earth orbit and carry a payload of microgravity experiments. The spacecraft uses Packet Telemetry, which is downlinked during the passes over the ground station in Maspalomas, Canary Islands. The payload data packets are transmitted to the Control Centre at the European Space Operations Centre (ESOC), Darmstadt after passes, where they are stored in history files. The packet telemetry standard allows separate packet identifiers for each experiment, so the data packets from each experiment are stored on separate files ordered according to the on-board time at which they were generated. The experiment owners are Principal Investigators (PIs) who work at their own home institutes and are generally not present at the control centre. They send detailed requests for operation of their instruments to the control centre, which plans them into a command schedule and then acquires and stores the resulting experimental data. Instead of delivering the data on magnetic tape or other transportable media, the data is made available to the users electronically in the so-called EURECA Data Disposition system (DDS). The host computers at the control centre are DEC/VMS and the data distribution takes place using a number of communications protocols (DECNET, FTAM, FTP and KERMIT).

The DDS delivers data to the users by file transfer. Each transfer takes place as a result of a PI request for data, which is itself a file transfer from the PI to the DDS. Each request results in the supply of one file of data. Thus if a PI requires a set of files, he issues a separate request for each one. Each PI can request either a list of all available data for his own instrument (i.e., a catalogue) or files of telemetry data packets produced by that instrument. He may also request ancillary data or any data which is deemed to be made available to the whole user community. Data Transfer Requests are in the form of an ASCII file. The details of these requests are not of interest in this context, since they are not in SFDU format.

The DDS responds with:

- An acknowledgement of the request.
- A catalogue entry, giving identifying information about the file supplied; this will include source/type identification, the time period spanned by the file, the consolidation time for that data stream, etc.
- The requested data set.

This response is packaged in an SFDU as shown in Figure 8-1. This reflects the components mentioned above, there being three LVOs. Envelope packaging with length delimitation is used. The resulting SFDU has:

1. A class V LVO carrying an acknowledgement record in which is contained a detailed specification of the original request and the request status.
2. A class K LVO containing a catalogue entry for the requested data set.
3. A class I LVO containing the requested data set itself.

While the first two LVOs are of fixed length, the application data sets will vary in size even for a given instrument, since the number of consecutive packets collected to constitute a file will depend upon the actual duty cycle (operation time) of the instrument, and on the time span requested by the PI.

It is intended that data descriptions are provided in text form. That is, English language ASCII text descriptions are supplied to the Control Authority. They are supplied to the Control authority in simple DDUs, containing the relevant description in a DDR (a class D LVO), the approach being similar to that

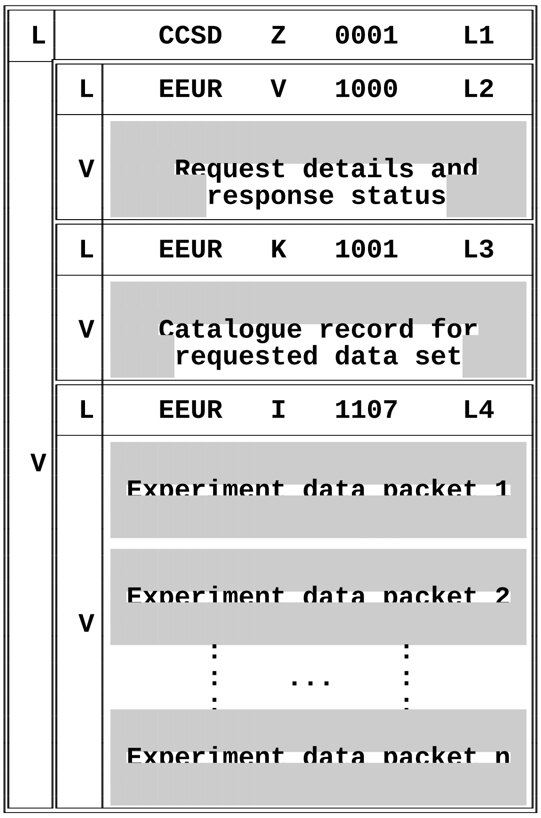


Figure 8-1: Schematic of Data Delivery Format for EURECA

described in the Data Description Package Registration example of Section 8.3.

Since the size of the class I LVO varies from request to request an alternative approach would have been to use marker pattern delimitation. This approach was not taken because an overriding concern was to minimise the complexity on the side of the users - while marker pattern delimitation can simplify matters for the data producer, it shifts work to the user (he has to reserve processing buffers and carry out marker searches).

The foregoing information sufficiently illustrates the principles. As examples, Table 8-1 shows the VALUE field of the Catalogue Record and Table 8-2 (overleaf) shows the VALUE field of the Acknowledgement Record. These records are comprised of ASCII text in fixed positions in the VALUE field. To ease readability, carriage return, line feed (CR/LF) pairs are interspersed throughout the text to break it into conveniently readable blocks when ingested into a word processor or text editor. All of the fields use ASCII characters to convey the data and are left justified within the field, if any of a field is not completely filled by the ASCII string, then it is padded by spaces. In the third column of Table 8-1 and Table 8-2, the underline symbol, "_", represents a space, a CR represents a carriage return and LF represents a line feed.

Field	Length in bytes	Notes
Screen format field 1	2	CR,LF
Spacecraft name	8	eg.EURECA__
Data Identifier	5	eg.ORA1_
Data type	5	eg.EXP__
ADID	9	eg.EEUR1001_
First packet time	24	DD-MMM-YYYY:HH:MM:SS.CC_
Last packet time	24	DD-MMM-YYYY:HH:MM:SS.CC_
Screen format field 2	2	CR,LF
Consolidated time	24	DD-MMM-YYYY:HH:MM:SS.CC_
Number of packets	9	Decimal ASCII string, no leading zeros
Spare	14	ASCII string
Screen format field 3	2	CR,LF

Table 8-1: VALUE field of Catalogue Record in EURECA DDS

In the International Solar Terrestrial Programme (ISTP) some approaches are being taken which are very similar to the EURECA DDS. The ISTP comprises the GEOTAIL, WIND and POLAR missions of NASA, the CLUSTER mission of ESA, and a NASA-ESA collaborative mission SOHO. Developed quite independently of the EURECA work described above, the NASA ISTP Central Data Handling Facility (CDHF) provides an interactive interface to each investigator’s Remote Data Analysis Facilities. The CDHF provides a more comprehensive set of services than the EURECA DDS. For example, it covers distribution of Key Parameter files and software as well as level zero data (equivalent to the EURECA experiment packets), its similarities to that work lie in the packaging of data files in an SFDU using length delimitation and a relatively simple structure consisting of content identifier (a class K LVO) followed by the data files themselves packaged in a class I LVO.

Field	Length in bytes	Notes
Screen format field 1	2	CR,LF
Spacecraft name	8	EURECA__
Data Source	5	eg. ORA1_
Data Type	5	eg. HK1__
Screen format field 2	2	CR,LF
Request filename	80	ASCII string
Screen format field 3	2	CR,LF
/BEFORE specified	24	From command line
/BEFORE expanded	24	DD-MMM-YYYY:HH:MM:SS.CC_
Screen format field 4	2	CR,LF
/SINCE specified	24	From command line
/SINCE expanded	24	DD-MMM-YYYY:HH:MM:SS.CC_
Screen format field 5	2	CR,LF
Target specified	80	ASCII string
Screen format field 6	2	CR,LF
Processed at	24	DD-MMM-YYYY:HH:MM:SS.CC_
Computer	10	eg. VAX_8750__
Operating system	6	eg. VMS___
Version	6	eg. 5.2__
Screen format field 7	2	CR,LF
Error message	80	ASCII string
Screen format field 8	2	CR,LF
Comments/Spare	80	ASCII string
Screen format field 9	2	CR,LF

Table 8-2: VALUE field of Acknowledgement Record in EURECA DDS

For the Cluster mission, a similar set-up to the EURECA DDS is foreseen. This interactive system is planned to be complemented by delivery of complete sets of mission data to all investigators on a transportable medium, probably optical disk. These data sets will also be packaged within SFDUs in a similar fashion to those delivered interactively via data lines.

8.2 Halley Watch

This example is derived from the experience of taking approximately 5,000 files of data acquired during spacecraft encounters with comet Halley and putting them on a CD-ROM for widespread distribution to users. This example is not identical to any individual Halley Watch CD-ROM and some unessential detail has been omitted for clarity and conciseness. However it captures the essence of SFDU usage in Halley Watch.

SFDU techniques are used to label all the data files (remotely if appropriate) and to create a sequential view of the resulting data product. This allows standard SFDU processing software to present a view of the CD-ROM content to users, and to allow the users to ask for and get the files and associated file descriptions, in any desired order, as required.

The nature of the data files varies widely, ranging from large images in binary to small ASCII text files in the form of tables. Users range from those with little resources and working solely in a PC environment to large research organisations with many investigators and corresponding support. The distribution medium is CD-ROM using the ISO 9660 volume and file structure standard. This standard provides a hierarchical directory structure and access to associated files, but does not provide standard attributes recognisable across different hardware and operating systems, that would allow one to indicate the type of a particular file, such as an ASCII text file. Thus in ISO 9660, all files are viewed as binary. The user may access certain parts of the data (browse text files) using a text editor or may use SFDU/LVO parsing and presentation software.

SFDU LABELs for automated access appear in a file called **`VOLDESC.SFD`** which is in the root directory of the random access medium. The overall structure of the VOLDESC file is shown in Figure 8-2, which consists of an SFDU which contains:

1. Header LVOs.

2. A further SFDU containing:

   - Description Data Units (DDUs)

   - Application Data Units (ADUs)

   All these DDUs and ADUs use referencing techniques to point to the relevant data objects stored elsewhere on the CD-ROM, i.e., external to the VOLDESC.SFD SFDU file.

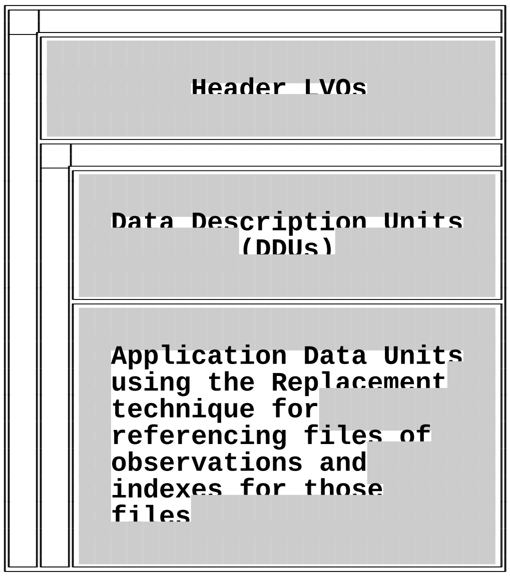

Header LVOs
Data Description Units (DDUs)
Application Data Units using the Replacement technique for referencing files of observations and indexes for those files

Figure 8-2: Overall Structure of VOLDESC.SFD SFDU

Table 8-3 (overleaf) lists the contents of the VOLDESC.SFD SFDU in more detail. Members (second column) whose names are followed with an asterisk are not actually present in the VOLDESC.SFD file, but are logically incorporated by the Replacement packaging technique, using LVOs of Class ID = R.

The application data of interest can be classified into three different common formats:

- Observations Type X (data of primary interest)
- Observations Type Y (other data of primary interest)
- Observations Type Z (data of secondary interest)

For simplicity, the examples given here only refer to Observations of Type X, so this will be the only type referred to in the remaining text.

Category	Members	Definition	Contents
Header LVOs	VPI	Volume Preparation Information	Who prepared the product, when, what media type, operating system, etc.
	PA	Product Attributes	Title, author, time coverage of product, date of publication, etc. (i.e. typical product cataloguing attributes)
	ID*	Introductory Description	A READ.ME file giving an overview of the CD-ROM, including its directory organisation
	DO	Detailed Overview	Text LVO giving an overview of the data source(s), reasons for collection, types of data, processing history, data quality, expected uses for the data, etc.
Data Descriptions	VPIDD*	VPI Data Description	Description of Volume Preparation Information in PVL
	PADD*	DD for Product Attributes	Data description of Product Attributes VALUE field in PVL
	OTXDD*	Observation Type X Data Description	An English language text description of the format and meaning of the data fields of the common format used for all OTX data objects
	IIDD*	DD for Inventory & Index	A data description for Product Inventory (PI) and the Indexes (OTXI1, OTXI2,..). These all have the same tabular structure.
Data Inventory and Indexes	PI*	Product Inventory	Inventory of all files listed according to some key such as time
Index and References to X Type Objects	OTXI1*	Index 1 for OTX files	OTX Index 1 (OTXI1) file. This file lists all OTX objects in order of a key '1' (e.g. time,) giving location in the product and other attributes.
	OTXI2*	Index 2	Similar to OTXI1; OTX Index 2 (OTXI2) file which lists all OTX objects in order of a key '2' (e.g. spatial location)
	OTX*	Observation Type X files	Contains all the data in X type format and meaning

Table 8-3: VOLDESC.SFD File Contents

Figure 8-3 (overleaf) shows the product in more detail. For brevity **`REFERENCETYPE`** statements are not explicitly shown. However in practice each class R LVO includes a **`REFERENCETYPE`** statement defining the reference environment as **`$CCSDS1`**. In this environment an asterisk (**`*`**) is used as a wildcard to refer to a collection of files. Thus **`OTX*.*`** refers to all the files containing OTX at the start of their filename, and the preceding SFDU LABEL logically labels each of these files, thereby forming logical LVOs. Note that the filenames used in the SFDU product are in the **`$CCSDS1`** format, and the filenames of the files that are actually accessed, pointed to on the right had side of the diagram, are in their native VAX VMS format for addressing a CD-ROM.

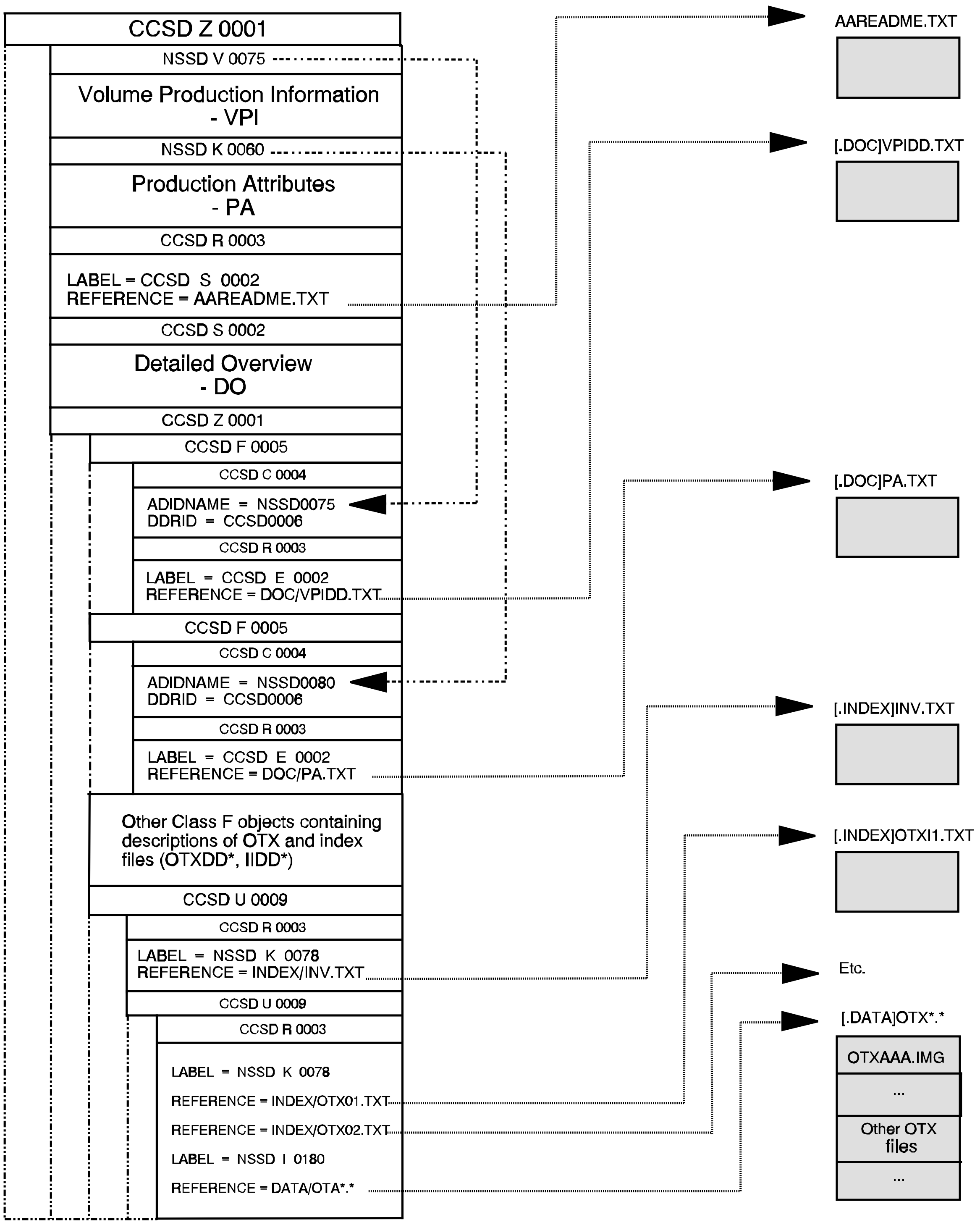


Figure 8-3: Detailed Schematic of VOLDESC.SFD File

8.3 Data Description Registration and Dissemination

In this example, SFDUs are used in the packaging of data descriptions. A Registration Package (RP) is used in the submission of all the information necessary to register data descriptions with a Control Authority Office. Each RP must therefore contain all the information which goes into the DDUs needed for processing SFDU data products. This example is drawn from experience gained in the operation of two NASA CAOs: the primary NASA CAO at NSSDC and a CAO operated by the UARS satellite project. It highlights the use of the SFDU packaging techniques for both the registration and dissemination functions. It also shows the use of SFDUs in a text environment.

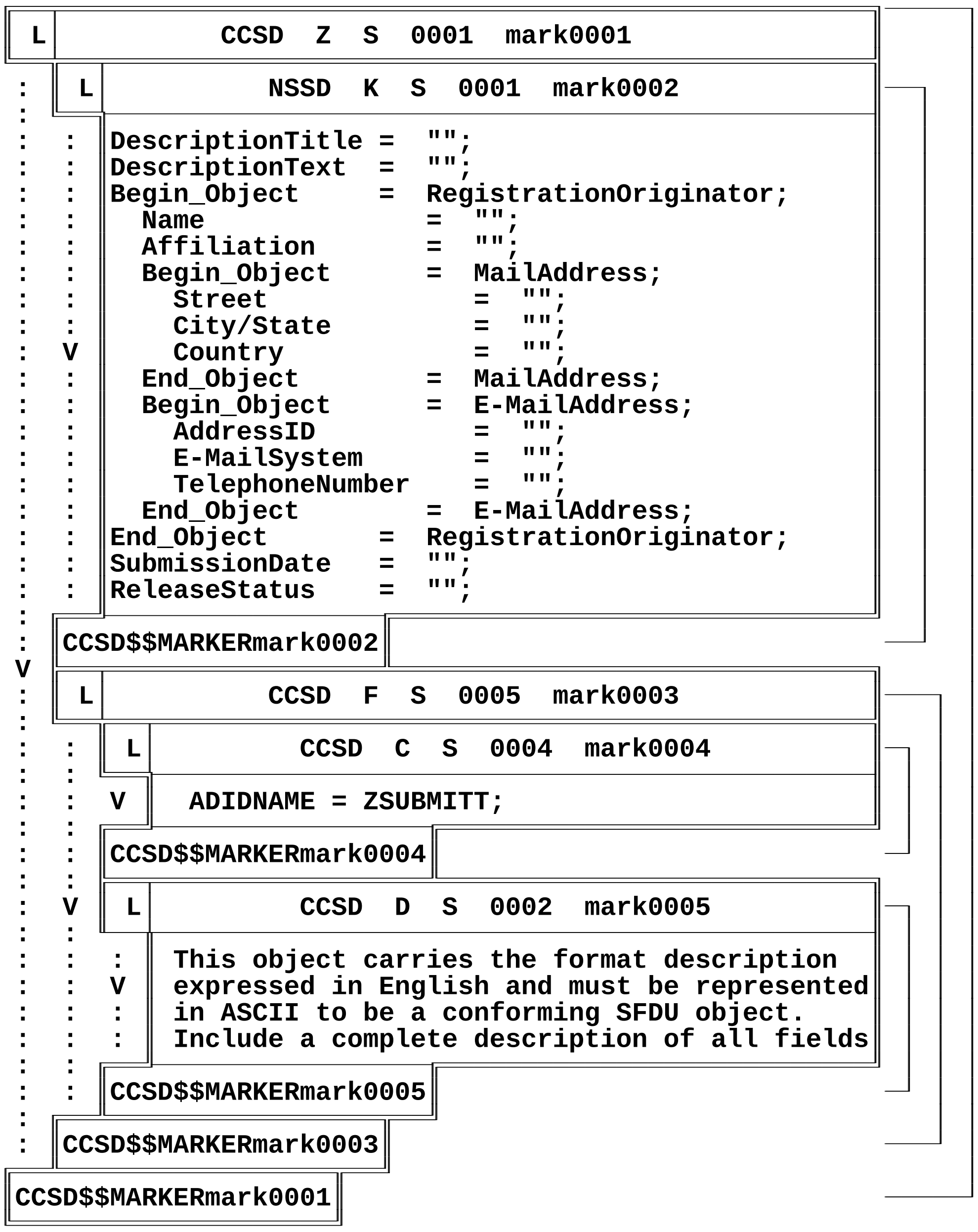
A data producer has defined a data structure (or data object) that will occur frequently in a data stream destined for many of his colleagues. This data object will carry instrument values of magnetic field observations taken each minute. The data producer has created an English language text description of the format of the data object using his local word processor, and desires to register this description with a CAO so that he need not provide it to each recipient of his data objects.

His local CAO provides him via electronic mail with a template for a Registration Package which may be submitted via electronic mail. This template is in SFDU form, and expressed in ASCII and is therefore a conforming SFDU product. The template is shown schematically in Figure 8-4 (overleaf). It uses marker pattern delimitation throughout and thus is completely suitable for handling in a word processor. The data producer accordingly imports it into his word processor.

This template consists of an EDU (class Z LVO) containing a catalogue object (class K LVO) expressed in PVL (see Reference [2]) followed by a DDU (class F LVO). The catalogue object includes the minimum attributes required (not all shown here) by all CAOs (see Reference [3]), and the DDU contains its instance identifier (class C LVO) and the data description (class D LVO) itself. For space reasons, explanatory comments in the PVL are omitted. This template is quite simple and does not have a separate dictionary object (class E LVO), nor does it suggest the data producer should include previously registered information as may be done using the class C LVO.

The data producer transmits an ASCII text file of the completed template to the CAO via e-mail. The CAO verifies that the identification information in the catalogue object is complete and that the SFDU packaging conforms to the standard. Following this, the CAO assigns the ADID (for example, NSSD0134) to this information and updates the **`ADIDNAME`** statement in the class C LVO. The resulting Data Description Package is stored by the CAO using its internal mechanisms. The registered ADID of NSSD0134 is sent to the data producer along with a copy of the registered information in the form of the Data Description Package. The text file received by the Data Producer from the CAO is shown in Figure 8-5 (page 60). For brevity, only extracts of the contents of the various component objects are shown. This figure shows:

- Examples of completion of the template both by the data producer and the CAO.
- That the LVO headers appear as 20 octet character strings in the text file.
- That the "lines" of PVL reflect the line delimitation techniques of the word processor used. This may differ between the word processor used by the data producer and the CAO. For example, the former's word processor may delimit lines by CR/LF whereas the latter uses CR only. This does not matter, provided the import packages make the correct transformations of the line delimiters. Of course the length of the VALUE field may change, but with the marker delimitation technique this does not matter.



```
L   CCSD  Z  S  0001  mark0001
  L   NSSD  K  S  0001  mark0002
    DescriptionTitle =  "";
    DescriptionText  =  "";
    Begin_Object     =  RegistrationOriginator;
      Name              =  "";
      Affiliation       =  "";
      Begin_Object      =  MailAddress;
        Street             =  "";
        City/State         =  "";
        Country            =  "";
      End_Object        =  MailAddress;
      Begin_Object      =  E-MailAddress;
        AddressID          =  "";
        E-MailSystem       =  "";
        TelephoneNumber    =  "";
      End_Object        =  E-MailAddress;
    End_Object       =  RegistrationOriginator;
    SubmissionDate   =  "";
    ReleaseStatus    =  "";
  CCSD$$MARKERmark0002
  L   CCSD  F  S  0005  mark0003
    L   CCSD  C  S  0004  mark0004
    V    ADIDNAME = ZSUBMITT;
    CCSD$$MARKERmark0004
    L   CCSD  D  S  0002  mark0005
        This object carries the format description
        expressed in English and must be represented
        in ASCII to be a conforming SFDU object.
        Include a complete description of all fields
    CCSD$$MARKERmark0005
  CCSD$$MARKERmark0003
CCSD$$MARKERmark0001
```

Figure 8-4: Schematic for a Data Description Registration Package Template

```
CCSD3ZS00001mark0001NSSD3KS00020mark0002

/* This object carries identification information conforming to        *
 * description NSSD0020 at the NASA/NSSDC Control Authority. NSSD0020   *
 * requires the use of PVL and a minimum set of parameter names.        *
 * Recall that values needing spaces are to be enclosed in quotes.      */

SubmissionDate     = 1991-01-24 ;            /* YYYY-MM-DD form        */
RevisionNumber     = 1 ;                     /* 1 is initial submission*/
DescriptionTitle   = "Magnetometer:Case 1" ;  /* Brief title           */
DescriptionText    = "The associated description describes the
   magnetometer data objects known as Case 1.  It covers three
   components in spacecraft coordinates and the field magnitude." ;
Begin_Object       =   RegistrationOriginator ;
   Name               =  "John Doe" ;         /* Person or Organization */
   Affiliation        =  NASA/GSFC  ;         /* Parent Organization    */
   Begin_Object       =   MailAddress
      Street              =  "Greenbelt Rd" ;
      City/State          =  "Greenbelt, MD" ;

  (ETC.......)

/*  This completes the identification information for the submitted     *
 *  Registration Package                                                */

CCSD$$MARKERmark0002CCSD3FS00005mark0003CCSD3C00004mark0004

/* This object carries the registered identification (ADIDNAME) of then *
 * description after update by the CA.  Prior to this it should carry a *
 * locally defined identifier beginning with a "Z".                     */

ADIDNAME   =   NSSD0134 ;

CCSD$$MARKERmark0004CCSD3DS00002mark0005

  (Definition of magnetometer data object format and meaning)

CCSD$$MARKERmark0005CCSD$$MARKERmark0003CCSD$$MARKERmark0001
```

Figure 8-5: Data Description Package in SFDU Form as Received from Control Authority

8.4 Data Description Language Usage for Data Description Record Writing

The SFDU concept provides a mechanism to associate application data with their description. Application data are contained in the VALUE field of a Simple LVO. Each of these Simple LVOs identifies a DDR (by means of its ADID) that is used to describe the application data that they contain.

The DDRs could be written either, in a natural language document that is only understandable by a human and not a computer (this is the more usual and traditional approach), or in a formal syntax. The advantages of writing the DDR in a formal syntax are:

1. There is automatic access to the described data. This access can be performed:

    - Either, by means of an "interpreter/parser" that interprets the DDR to parse the data elements and then delivers them to the application.

    - Or, by using the DDR directly in the application (i.e., the DDL is written in a programming language and then directly incorporated within the application, and compiled with the application software).

2. A computer can be used to assist in the design and writing of the DDR.

3. It is possible to automatically generate the actual data in a VALUE field from the syntax described in the DDR. This ensures consistency between the data and the data description.

Further CCSDS Recommendations will approve some DDLs for this purpose. There follows an example to illustrate how this application data description could be managed.

The example used here, DORIS telemetry (the layout of which is shown in Figure 8-6 (overleaf)), is transmitted to the ground station in a format that begins with a synchronisation bit pattern (hexadecimal value CD04) and the structure of which depends on the type of frame. The type of frame (measurement, housekeeping 1 or 2, jamming, normal or incident dump) is given by the frame identifier.

A frame has the following mandatory fields:

- Synchronisation bit pattern;
- The type of frame identifier;
- The least significant bits of the date.

All other fields in the frame are optional depending on the value of the frame identifier. In particular the most significant bits of the date are given by the house-keeping frame.

The DDR provides:

1. A logical definition of each elementary or composite field;

2. Information about the physical representation, such as:

   - The location of the elementary fields within the composite fields.
   - The bit pattern associated with each value of an elementary field (e.g., **`0000`** associated with **`MEASUREMENT`**, **`1010`** associated with **`HOUSEKEEPING 2`**, . . .).

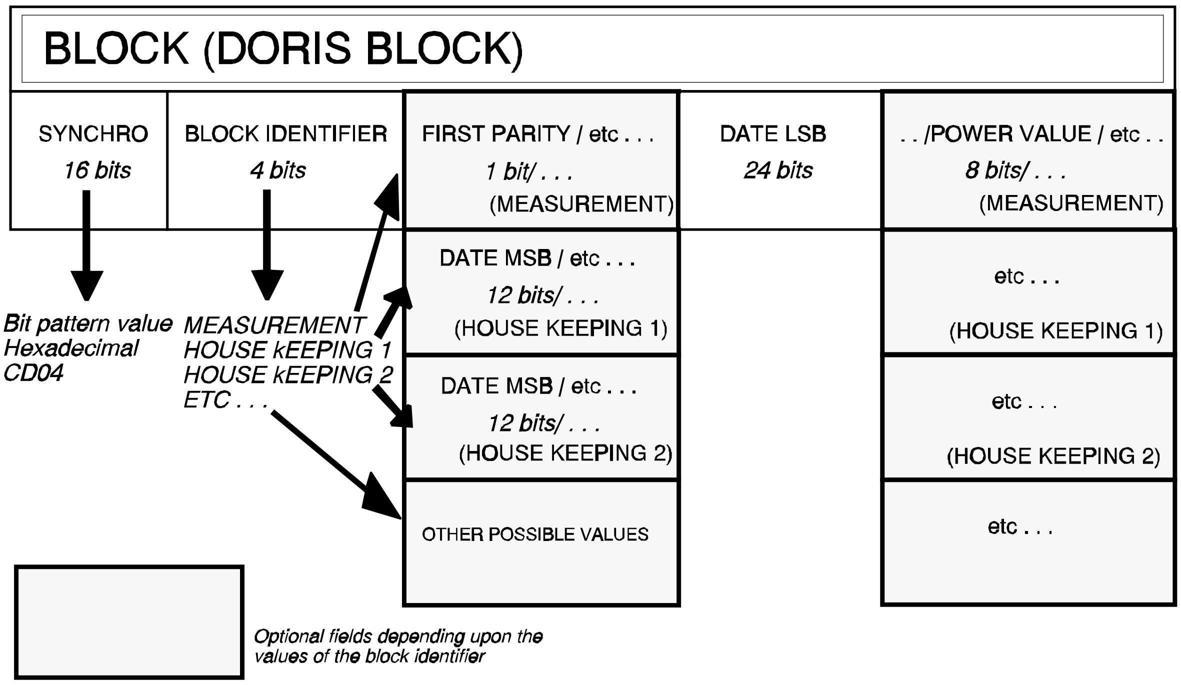

BLOCK (DORIS BLOCK)
SYNCHRO 16 bits
BLOCK IDENTIFIER 4 bits
FIRST PARITY / etc . . . 1 bit/ . . . (MEASUREMENT)
DATE LSB 24 bits
. . /POWER VALUE / etc . . 8 bits/ . . . (MEASUREMENT)
DATE MSB / etc . . . 12 bits/ . . . (HOUSE KEEPING 1)
etc . . . (HOUSE KEEPING 1)
DATE MSB / etc . . . 12 bits/ . . . (HOUSE KEEPING 2)
etc . . . (HOUSE KEEPING 2)
OTHER POSSIBLE VALUES
etc . . .
Bit pattern value Hexadecimal CD04
MEASUREMENT HOUSE kEEPING 1 HOUSE kEEPING 2 ETC . . .
Optional fields depending upon the values of the block identifier

Figure 8-6: Schematic Layout of DORIS Telemetry

The DDR for DORIS telemetry (shown in Figure 8-7 (page 64) and Figure 8-8 (page 65)) shows how to use some features of the Ada programming language to describe a telemetry frame. The choice of DORIS telemetry provides typical examples of scalar data and structured data (with mandatory and optional structural parts).

The full description is too long to show in this example, therefore, some fields have been suppressed and replaced by "**`......../........`**".

Figure 8-7 and Figure 8-8 show the DDR written in Ada describing the DORIS telemetry frame. This DDR specifies both the logical definition of the telemetry frame and the physical representation of every field in the frame. For the reader unfamiliar with Ada, these figures will be explained in more detail.

Figure 8-7 shows an Ada declarative section which declares the type of each elementary field in the frame (according to Ada, every variable must have a corresponding type explicitly assigned). In this figure, is found:

- A list of the "enumerated types" used (each corresponds to a list of discrete values). The first enumerated type declared (the statement "**`type KIND_OF_BLOCK ...`**") defines the list of enumerated values **`MEASUREMENT, HOUSE_KEEPING_1, ...`**. The following statement ("**`for`**

KIND_OF_BLOCK use ...") explicitly associates each of the previous enumerated values with a corresponding 4 bit pattern which will really be found in the data. The same approach is used for the data type declarations of **PARITY**, **KIND_OF_BEACON** and **SYNCHRO_PATTERN**. Note that for **SYNCHRO_PATTERN**, there is only one bit pattern corresponding to the hexadecimal number CD04.

- A list of the "numeric types" used. The first numeric type declared (the statement "**type ON_BOARD_CLOCK_PIP is range 0 .. 16777215**") specifies an integer value type which has a decimal range of 0 to 16777215. The following statement ("**for ON_BOARD_CLOCK_PIP'size use 24**") declares the integer as a 24 bit field. Again this is repeated for the other numeric types **INTEGER_12_BITS**, **PRESSURE**, **POWER** and **DIGIT**.

Figure 8-8 which is the continuation of Figure 8-7 shows two sets of Ada declarations:

- The "logical description" shows the structured type **DORIS_BLOCK** which declares between the Ada keywords "**record**"and "**end record**" a number of elementary fields. Note that the first statement declares a structure parameterised by the variable **BLOCK_IDENTIFIER** which is itself declared of type **KIND_OF_BLOCK** in Figure 8-7. The variables **SYNCHRO** and **DATE_LSB** are declared also in Figure 8-7 of types **SYNCHRO_PATTERN** and **ON_BOARD_CLOCK_PIP** respectively. There follows the variant part of the **DORIS_BLOCK** structure which depends on the value of **BLOCK_IDENTIFIER**, and is introduced by the statement "**case BLOCK_IDENTIFIER is**", and terminates with the statement "**end case**". For each possible value of **BLOCK_IDENTIFIER** (the statement "**when...=>**"), the list of the variables corresponding to the fields appearing for that case are defined.

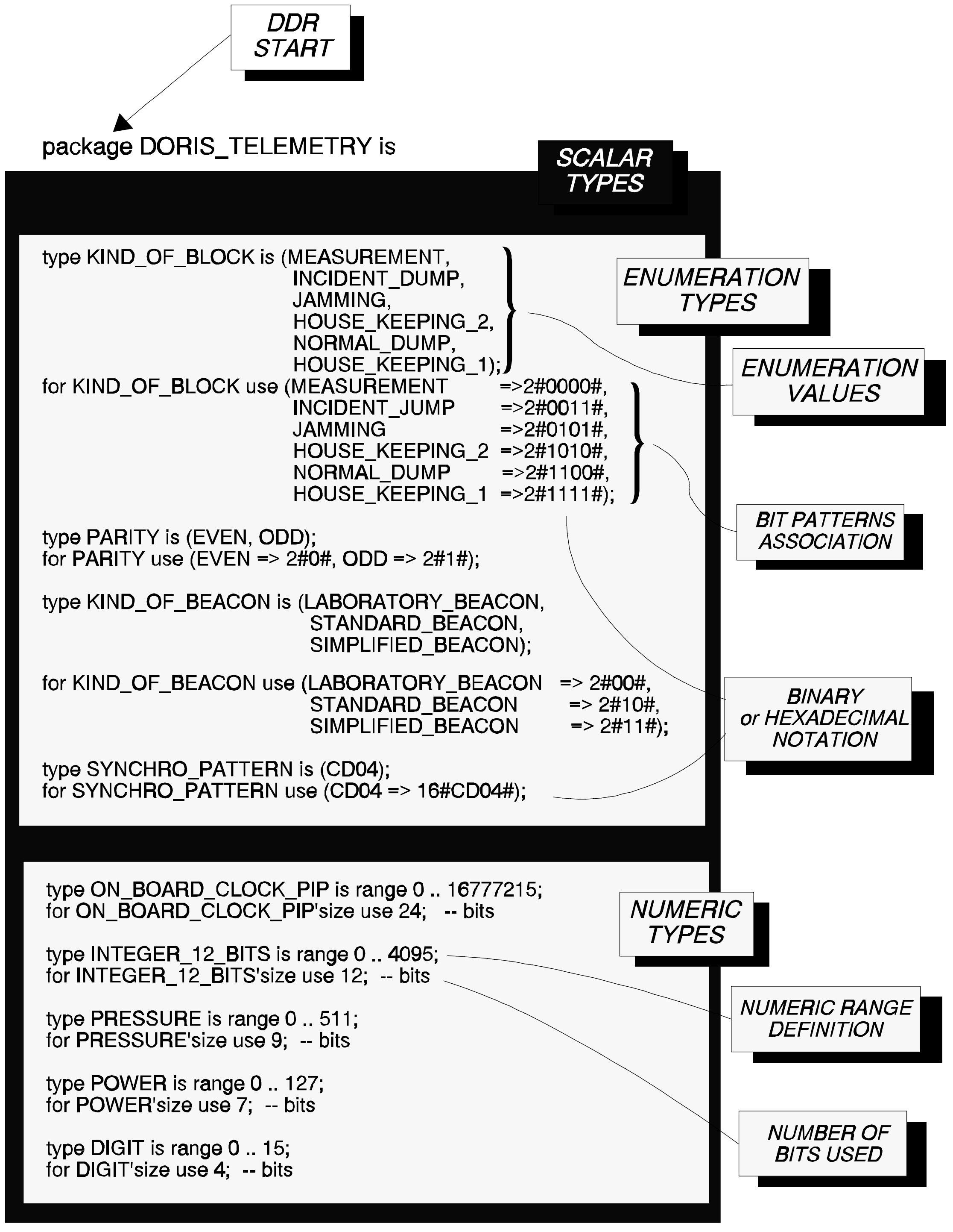
This description does not yet indicate how these different elementary fields are physically located in the telemetry frame. The "physical location of fields" section has the role of indicating the physical location of each elementary field in the **DORIS_BLOCK** description. The statement "**for DORIS_BLOCK use**" introduces a list starting with the keyword "**record**" which assigns every variable previously declared a fixed physical location in the telemetry frame. For example, the statement "**BLOCK_IDENTIFIER at 1*WORD_16_BITS range 0 .. 3 ;**" indicates that the field **BLOCK_IDENTIFIER** starts at the second 16 bit word (word number 1) from the beginning of the **DORIS_BLOCK** structure and occupies bits 0 to 3 (bit 0 being the most significant bit of the 16 bit word). This technique is used for each of the elementary fields.

This example shows how an existing powerful language can be used to:

- Define the logical structure with optional parts depending on the value stored in a previous mandatory field.

- Define the physical representation of the elementary fields of a structure.

It should be noted that in using Ada as a data description language as demonstrated above, it is not necessary to write the application in Ada or use an Ada compiler. In fact, a knowledge of Ada is not required at all, if the a suitable user friendly DDR generation tool is used. Only the data declaration aspects of the Ada language are being used here and therefore an interpreter for only this part of the language is required. This interpreter could be written in one of many programming languages, as can the application using the descriptions.

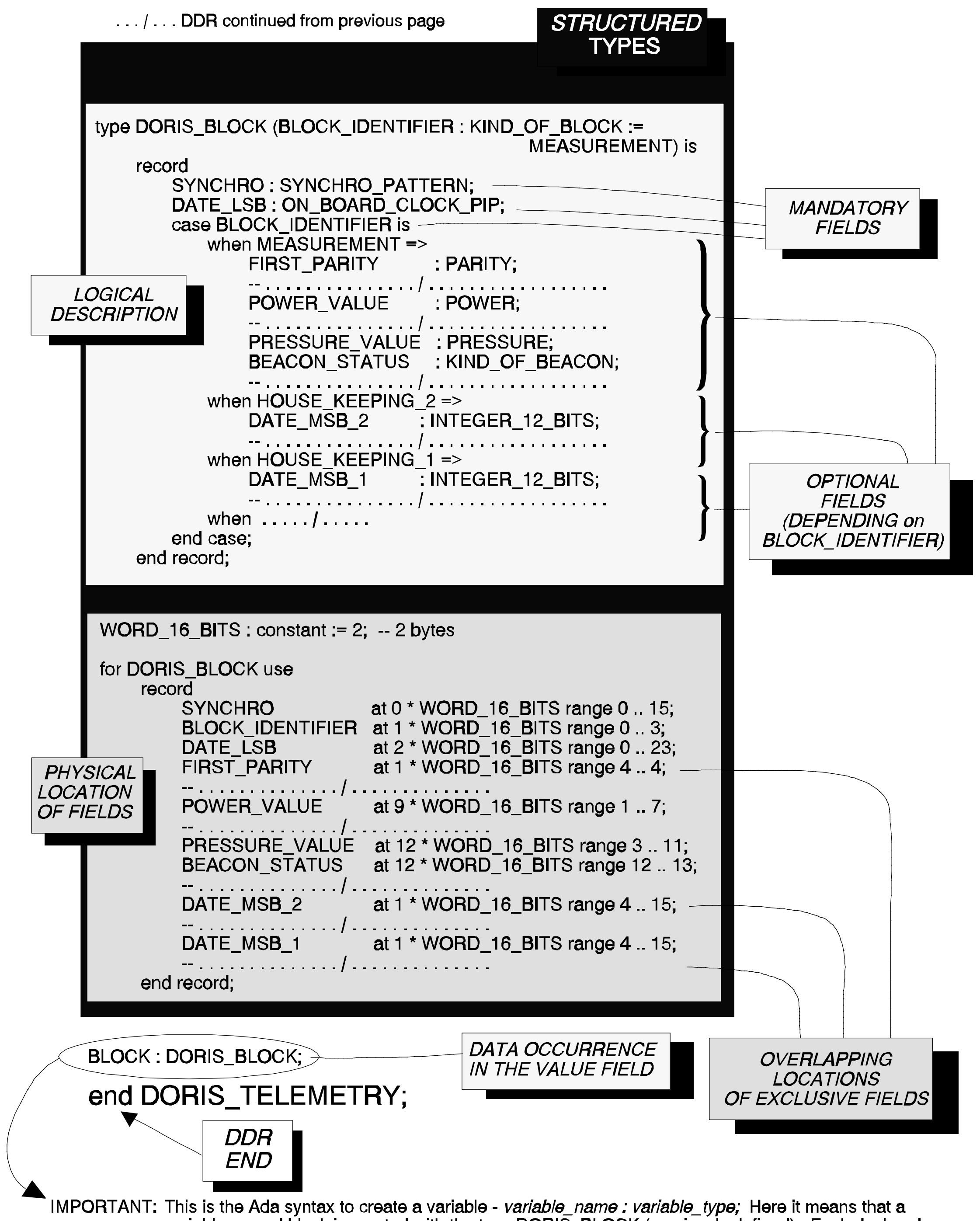


```
package DORIS_TELEMETRY is

type KIND_OF_BLOCK is (MEASUREMENT,
                       INCIDENT_DUMP,
                       JAMMING,
                       HOUSE_KEEPING_2,
                       NORMAL_DUMP,
                       HOUSE_KEEPING_1);
for KIND_OF_BLOCK use (MEASUREMENT      =>2#0000#,
                       INCIDENT_JUMP    =>2#0011#,
                       JAMMING          =>2#0101#,
                       HOUSE_KEEPING_2  =>2#1010#,
                       NORMAL_DUMP      =>2#1100#,
                       HOUSE_KEEPING_1  =>2#1111#);

type PARITY is (EVEN, ODD);
for PARITY use (EVEN => 2#0#, ODD => 2#1#);

type KIND_OF_BEACON is (LABORATORY_BEACON,
                        STANDARD_BEACON,
                        SIMPLIFIED_BEACON);

for KIND_OF_BEACON use (LABORATORY_BEACON  => 2#00#,
                        STANDARD_BEACON    => 2#10#,
                        SIMPLIFIED_BEACON  => 2#11#);

type SYNCHRO_PATTERN is (CD04);
for SYNCHRO_PATTERN use (CD04 => 16#CD04#);

type ON_BOARD_CLOCK_PIP is range 0 .. 16777215;
for ON_BOARD_CLOCK_PIP'size use 24;  -- bits

type INTEGER_12_BITS is range 0 .. 4095;
for INTEGER_12_BITS'size use 12;  -- bits

type PRESSURE is range 0 .. 511;
for PRESSURE'size use 9;  -- bits

type POWER is range 0 .. 127;
for POWER'size use 7;  -- bits

type DIGIT is range 0 .. 15;
for DIGIT'size use 4;  -- bits
```

. . . / . . . DDR continued on next page

Figure 8-7: Example of an Ada DDR Describing the DORIS Equipment Telemetry

. . . / . . . DDR continued from previous page


STRUCTURED TYPES
MANDATORY FIELDS
LOGICAL DESCRIPTION
OPTIONAL FIELDS (DEPENDING on BLOCK_IDENTIFIER)
PHYSICAL LOCATION OF FIELDS
DATA OCCURRENCE IN THE VALUE FIELD
OVERLAPPING LOCATIONS OF EXCLUSIVE FIELDS
DDR END

```
type DORIS_BLOCK (BLOCK_IDENTIFIER : KIND_OF_BLOCK :=
                                                  MEASUREMENT) is
    record
        SYNCHRO : SYNCHRO_PATTERN;
        DATE_LSB : ON_BOARD_CLOCK_PIP;
        case BLOCK_IDENTIFIER is
            when MEASUREMENT =>
                FIRST_PARITY        : PARITY;
                -- . . . . . . . . . . . . . . . / . . . . . . . . . . . . . . . . .
                POWER_VALUE         : POWER;
                -- . . . . . . . . . . . . . . . / . . . . . . . . . . . . . . . . .
                PRESSURE_VALUE   : PRESSURE;
                BEACON_STATUS    : KIND_OF_BEACON;
                -- . . . . . . . . . . . . . . . / . . . . . . . . . . . . . . . . .
            when HOUSE_KEEPING_2 =>
                DATE_MSB_2        : INTEGER_12_BITS;
                -- . . . . . . . . . . . . . . . / . . . . . . . . . . . . . . . . .
            when HOUSE_KEEPING_1 =>
                DATE_MSB_1        : INTEGER_12_BITS;
                -- . . . . . . . . . . . . . . . / . . . . . . . . . . . . . . . . .
            when . . . . . / . . . . .
        end case;
    end record;
```

```
WORD_16_BITS : constant := 2;  -- 2 bytes

for DORIS_BLOCK use
    record
        SYNCHRO              at 0 * WORD_16_BITS range 0 .. 15;
        BLOCK_IDENTIFIER  at 1 * WORD_16_BITS range 0 .. 3;
        DATE_LSB             at 2 * WORD_16_BITS range 0 .. 23;
        FIRST_PARITY        at 1 * WORD_16_BITS range 4 .. 4;
        -- . . . . . . . . . . . . . / . . . . . . . . . . . . .
        POWER_VALUE         at 9 * WORD_16_BITS range 1 .. 7;
        -- . . . . . . . . . . . . . / . . . . . . . . . . . . .
        PRESSURE_VALUE    at 12 * WORD_16_BITS range 3 .. 11;
        BEACON_STATUS      at 12 * WORD_16_BITS range 12 .. 13;
        -- . . . . . . . . . . . . . / . . . . . . . . . . . . .
        DATE_MSB_2          at 1 * WORD_16_BITS range 4 .. 15;
        -- . . . . . . . . . . . . . / . . . . . . . . . . . . .
        DATE_MSB_1          at 1 * WORD_16_BITS range 4 .. 15;
        -- . . . . . . . . . . . . . / . . . . . . . . . . . . .
    end record;
```

```
BLOCK : DORIS_BLOCK;

end DORIS_TELEMETRY;
```

IMPORTANT: This is the Ada syntax to create a variable - *variable_name : variable_type;* Here it means that a variable named block is created with the type DORIS_BLOCK (previously defined). Each declared variable in the Ada DDR corresponds to a data occurrence in the described VALUE field. The type definitions are used to define the variables but they do not correspond to any data occurrences in the described VALUE field.

Figure 8-8: Example of an Ada DDR Describing the DORIS Equipment Telemetry (continued from previous page)

Annex A

STANDARD FORMATTED DATA UNIT ACRONYMS

Purpose:

This annex defines the acronyms which are used throughout this report to describe the concepts and elements of the Standard Formatted Data Unit.

Annex A: Acronyms

ADID	Authority Description Identifier
ADU	Application Data Unit
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
CAO	Control Authority Office
CA	Control Authority
CAID	Control Authority Identifier
CCSDS	Consultative Committee for Space Data Systems
DDID	Data Description Identifier
DDL	Data Description Language
DDR	Data Description Record
DDU	Description Data Unit
DED	Data Entity Dictionary
EOF	End of File
ID	Identifier
LVO	LABEL-VALUE-Object
MACAO	Member Agency Control Authority Office
PVL	Parameter Value Language
RA	Restricted ASCII
SFDU	Standard Formatted Data Unit

ANNEX B

STANDARD FORMATTED DATA UNIT GLOSSARY

Purpose:

This annex defines key terms which are used throughout this report to describe the concepts and elements of the Standard Formatted Data Unit.

Annex B: Glossary of Terms

Control Authority: An organisation under the auspices of CCSDS which supports the transfer and usage of SFDUs by providing operational services of registration, archiving and dissemination of data description data. It comprises:

- The CCSDS Secretariat supported by a CA Agent
- Member Agency Control Authority Offices (MACAOs)

Cross support: When one agency uses part of another agency's data system resources to complement its own system.

Data Description Language (DDL): A language for describing the logical representation of data.

Data Description Record (DDR) (also referred to as Data Description Record Object): A syntactic description for data entities.

Data Element: The smallest named item or items of data for a given application.

Data Entity: A named collection of data elements.

Data Entity Dictionary (DED): A collection of semantic definitions for data entities.

Data Object: A collection of data elements that are packaged for or by a specific application.

Data Product: A collection of one or more data objects.

Delimitation: The method of specifying the end of a block of data.

Instance: A specific occurrence of values of a data entity.

Instance-dependent metadata: metadata that is applicable to only one data instance. For example, catalogue data.

Instance-independent metadata: metadata that is applicable to many data instances. For example, a format description that is applicable to many data instances.

Metadata: Data about other data.

Member Agency Control Authority Office (MACAO): an individual CCSDS Member-Agency organisation which has accepted the operational responsibilities and constraints on CA operations. For any given Member Agency, there will in general be one primary MACAO and one or more descendent MACAOs.

Open system data interchange: The process of transferring data from one open system to another. An open system is one which uses publicly available formats and protocols, so that anyone can communicate with the open system by following the open system standards. It should be noted that open system does not imply an uncontrolled or unrestricted access to the data.

Reference environment: An environment in which the value of a REFERENCE statement is understood to give one or more locations at which external data objects begin.

Semantic information: Information associated with data that defines the meaning of the data.

Syntactic information: Information associated with data that defines the format of the data.

INDEX