

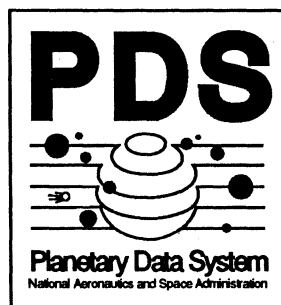
D-4683

# Standards for the Preparation & Interchange of Data Sets

T. Z. Martin, M. D. Martin, R. L. Davis, R. Mehlman  
M. Braun, M. Johnson

Version 1.1

October 3, 1988



Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, California



## TABLE OF CONTENTS

1. INTRODUCTION . . . . .	1-1
1.1. DOCUMENT OVERVIEW . . . . .	1-1
1.2. ACKNOWLEDGEMENTS . . . . .	1-2
1.3. REFERENCES . . . . .	1-2
2. DATA SET SUBMISSION PROCESS . . . . .	2-1
2.1. DATA SUPPLIERS . . . . .	2-1
2.2. SUPPORT FOR DATA SET PREPARATION . . . . .	2-1
2.3. PDS CONTACT PERSONNEL . . . . .	2-1
2.4. PROCEDURE FOR DATA SET SUBMISSION . . . . .	2-2
2.4.1. Estimation . . . . .	2-2
2.4.2. Specification . . . . .	2-2
2.4.3. Preparation . . . . .	2-2
2.4.4. Submission . . . . .	2-3
2.4.5. Review . . . . .	2-4
2.4.6. Rework . . . . .	2-4
2.4.7. Signoff . . . . .	2-4
3. DOCUMENTATION STANDARDS . . . . .	3-1
3.1. RECOMMENDED FORMAT . . . . .	3-1
3.2. EXAMPLES . . . . .	3-1
3.3. SOFTWARE DOCUMENTATION . . . . .	3-1
3.4. DOCUMENTATION CONTENTS . . . . .	3-2
4. CATALOG STANDARDS . . . . .	4-1
4.1. PDS SCIENCE CATALOG ARCHITECTURE . . . . .	4-1
4.2. ENTITY RELATIONSHIP MODEL . . . . .	4-2
5. PDS NOMENCLATURE STANDARDS . . . . .	5-1
5.1. PDS DATA NOMENCLATURE SYNTAX . . . . .	5-1
5.1.1. Construction of Terms . . . . .	5-2
5.1.2. Order of Terms within a Data Object Name . . . . .	5-2
5.1.3. Specifiers . . . . .	5-2
5.1.4. Descriptor Words . . . . .	5-2
5.1.5. Class Words . . . . .	5-3

## TABLE OF CONTENTS (CONT.)

5.1.6. Abbreviations . . . . .	5-3
5.1.6.1. Abbreviation of PDS Formal Data Object Names . . . . .	5-3
5.1.6.2. The Construction of Terse Data Element Names . . . . .	5-4
5.2. PDS FILE NAME SYNTAX . . . . .	5-5
5.2.1. Naming Rules . . . . .	5-5
5.2.1.1. Standard File Extensions . . . . .	5-5
5.3. DIRECTORY NAMING AND USAGE CONVENTIONS . . . . .	5-6
6. DATA FORMAT STANDARDS . . . . .	6-1
6.1. DATA FORMAT REPRESENTATION SYSTEMS . . . . .	6-1
6.1.1. Predefined Structures . . . . .	6-1
6.1.2. Keyword Structures . . . . .	6-1
6.2. PDS OBJECT DESCRIPTION LANGUAGE (ODL) . . . . .	6-2
6.2.1. ODL Objectives . . . . .	6-2
6.2.2. ODL Concept . . . . .	6-2
6.2.3. Data Object Description . . . . .	6-3
6.2.4. Data Objects Storage and Transportation . . . . .	6-3
6.2.5. Object Class Hierarchies and Inheritance . . . . .	6-5
6.2.6. Summary . . . . .	6-6
7. MISCELLANEOUS STANDARDS . . . . .	7-1
7.1. TIME . . . . .	7-1
7.1.1. Representations of Dates . . . . .	7-1
7.1.2. Representations of Times . . . . .	7-2
7.1.3. Dates and Times . . . . .	7-2
7.1.4. Periods of Time . . . . .	7-2
7.2. UNITS . . . . .	7-2
7.3. BINNING . . . . .	7-4
7.4. SOFTWARE DEVELOPMENT . . . . .	7-4
7.5. ANCILLARY GEOMETRIC INFORMATION (SPICE FILES) . . . . .	7-5
7.6. CARTOGRAPHIC DATA . . . . .	7-6
7.6.1. Inertial Reference Frame/Timetag/Units . . . . .	7-7
7.6.2. Spin Axes and Prime Meridians . . . . .	7-7

## TABLE OF CONTENTS (CONT.)

7.6.3. Reference Coordinates . . . . .	7-7
7.6.4. Reference Surface . . . . .	7-8
7.6.5. Map Resolution . . . . .	7-8
7.6.6. Documentation . . . . .	7-8
7.6.7. References . . . . .	7-8
8. TOOLS . . . . .	8-1
8.1. DATA SET SOFTWARE . . . . .	8-1
8.2. PDS LABEL SOFTWARE . . . . .	8-2

## APPENDICES

A. DATA INGESTION FROM PRESENT AND FUTURE MISSIONS . . . . .	A-1
A.1. DRAFT PDS MISSION DATA INTERFACE LIST . . . . .	A-1
A.2. WRITING THE PROJECT DATA MANAGEMENT PLAN . . . . .	A-1
A.3. MECHANISM FOR CHANGE OUTSIDE OF STEPS 1 AND 2 . . . . .	A-2
B. DATA RESTORATION PROCEDURES . . . . .	B-1
B.1. DATA RESTORATION PRIORITIZATION . . . . .	B-1
B.2. NASA REQUEST FOR DATA RESTORATION PROPOSALS . . . . .	B-1
B.3. SELECTION OF DATA RESTORATION PROPOSALS . . . . .	B-1
B.4. RESTORATION PROCESS . . . . .	B-1
C. WRITING CONVENTIONS AND DOCUMENT STANDARDS . . . . .	C-1
D. DATA SET DOCUMENTATION EXAMPLES . . . . .	D-1
D.1. PLASMA WAVE DOCUMENTATION EXAMPLE . . . . .	D-1
D.2. IRS DATA SET EXAMPLE . . . . .	D-9
E. ENTITY DEFINITIONS AND STRUCTURES . . . . .	E-1
E.1. COORDINATE SYSTEM INFORMATION . . . . .	E-2
E.2. DATA SET AND PRODUCT INFORMATION . . . . .	E-3
E.3. EARTH BASE INFORMATION . . . . .	E-4
E.4. EVENT INFORMATION . . . . .	E-4
E.5. INSTITUTION INFORMATION . . . . .	E-5
E.6. INSTRUMENT INFORMATION . . . . .	E-6
E.7. MISSION INFORMATION . . . . .	E-8
E.8. NODE INFORMATION . . . . .	E-9
E.9. PARAMETER INFORMATION . . . . .	E-10
E.10. PERSONNEL INFORMATION . . . . .	E-11
E.11. REFERENCE INFORMATION . . . . .	E-12
E.12. SOFTWARE INFORMATION . . . . .	E-12
E.13. SPACECRAFT INFORMATION . . . . .	E-13
E.14. TARGET BODY INFORMATION . . . . .	E-14
F. PDS CLASS AND DESCRIPTOR WORD DICTIONARY . . . . .	F-1
F.1. INTRODUCTION . . . . .	F-1
F.2. CLASS WORD DICTIONARY . . . . .	F-1

## APPENDICES (CONT.)

F.3. DESCRIPTOR WORD DICTIONARY . . . . .	F-2
F.4. PLURAL DESCRIPTOR WORDS . . . . .	F-9
F.5. IDENTIFIED NON-DESCRIPTOR WORDS . . . . .	F-10
G. PDS ABBREVIATION LIST . . . . .	G-1
H. STANDARD FORMATTED DATA UNITS . . . . .	H-1
H.1. INTRODUCTION . . . . .	H-1
H.2. INFORMATION TRANSFER . . . . .	H-1
H.3. DATA STRUCTURING . . . . .	H-1
H.4. DATA DEFINITION . . . . .	H-3
H.5. TERMINOLOGY . . . . .	H-3
H.6. ODL IMPLEMENTATION OF SFDU'S . . . . .	H-5
I. ODL IMPLEMENTATION AND SPECIFICATION . . . . .	I-1
I.1. IMPLEMENTATION OF PDS OBJECT DESCRIPTION LANGUAGE . . . . .	I-1
I.1.1. PDS Object Description Language . . . . .	I-1
I.1.2. Data Unit Labels . . . . .	I-3
I.1.2.1. SFDU Registration . . . . .	I-3
I.1.2.2. Data Unit Format Description . . . . .	I-4
I.1.2.3. Pointers to Objects . . . . .	I-4
I.1.2.4. Data Unit Content Description . . . . .	I-4
I.1.2.5. Object Descriptions . . . . .	I-5
I.1.2.6. END Statement . . . . .	I-5
I.1.3. Accessing Data Objects . . . . .	I-6
I.1.3.1. Describing Classes of Objects . . . . .	I-8
I.2. ODL SPECIFICATION . . . . .	I-10
I.2.1. Definitions . . . . .	I-10
I.2.2. Label Format . . . . .	I-11
I.2.3. Object Statements . . . . .	I-11
I.2.4. Attribute Assignment Statements . . . . .	I-11
I.2.4.1. Keyword . . . . .	I-12
I.2.4.2. Assignment Symbol . . . . .	I-12
I.2.4.3. Value . . . . .	I-12

## APPENDICES (CONT.)

I.2.5. Object Description Language Syntax Specification . . . . .	I-13
I.2.5.1. Basic Elements of the Language . . . . .	I-13
I.2.5.2. Lexical Elements . . . . .	I-13
I.2.5.3. Syntactic Elements . . . . .	I-14
I.2.5.4. Semantics . . . . .	I-15
J. DATA UNIT FORMATS . . . . .	J-1
J.1. RECOMMENDATIONS FOR USING RECORD FORMATS . . . . .	J-2
J.1.1. Fixed Length Record Formats . . . . .	J-2
J.1.2. Variable Record Formats . . . . .	J-3
J.1.3. Stream Record Formats . . . . .	J-3
J.1.4. Composite Files . . . . .	J-3
K. DATA OBJECT DESCRIPTIONS . . . . .	K-1
K.1. ELEMENTARY OBJECTS . . . . .	K-1
K.1.1. Specification Qualifiers . . . . .	K-2
K.1.1.1. Length Specification . . . . .	K-2
K.1.1.2. Binary Integer Specifications . . . . .	K-2
K.1.1.3. Signed Versus Unsigned . . . . .	K-2
K.1.1.4. Floating Point Formats . . . . .	K-2
K.1.1.5. Bit String Data . . . . .	K-3
K.1.2. Object Format Specifications . . . . .	K-3
K.1.3. Explicit Definitions of Elementary Objects . . . . .	K-4
K.2. AGGREGATE OBJECTS . . . . .	K-5
K.3. COMPOUND OBJECTS . . . . .	K-5
K.3.1. TEXT Object Format . . . . .	K-5
K.3.2. TABLE Object Format . . . . .	K-6
K.3.3. IMAGE Object Format . . . . .	K-8
K.3.4. QUBE Object Format . . . . .	K-10
K.3.4.1. Qube Terminology . . . . .	K-10
K.3.4.2. Label Keywords Describing the File Containing a Qube Object . . . . .	K-12
K.3.4.3. Basic Qube Object Keywords . . . . .	K-12
K.3.4.4. Qube Axis Keywords . . . . .	K-12



## APPENDICES (CONT.)

K.3.4.5. Qube Core Keywords . . . . .	K-14
K.3.4.6. Qube Prefix/Suffix Keywords . . . . .	K-15
K.3.4.7. Projection Keywords . . . . .	K-15
K.3.5. History Object Format . . . . .	K-16
K.3.5.1. History Entry . . . . .	K-16
K.3.5.2. Tree of Processes . . . . .	K-17
K.3.6. History Object Processing . . . . .	K-17
L. SAMPLE ODL LABELS . . . . .	L-1
L.1. SAMPLE TEXT FILE LABEL . . . . .	L-1
L.2. SAMPLE TABLE FILE LABELS . . . . .	L-2
L.3. SAMPLE IMAGE FILE LABELS . . . . .	L-10
L.4. SAMPLE CUBE FILE LABEL . . . . .	L-15
M. PDS CODING STANDARDS . . . . .	M-1
M.1. COMMENTS . . . . .	M-2
M.2. PROGRAMMING STYLE . . . . .	M-3
M.3. EXPLICIT TYPING . . . . .	M-4
M.4. NAMING CONVENTIONS . . . . .	M-5
M.5. LANGUAGE SPECIFIC PRACTICES . . . . .	M-5
M.5.1. PDS FORTRAN Coding Standards . . . . .	M-5
M.5.2. PDS C Coding Standards . . . . .	M-7
M.6. VALID CHARACTER SET . . . . .	M-9

## LIST OF FIGURES

1-1 Major Components of the SPIDS Document . . . . .	1-2
1-2 Standards in the SPIDS Document . . . . .	1-3
4-1 High-Level Catalog Schema . . . . .	4-3
H-1 Structured TLV Data Object . . . . .	H-2
H-2 Delimiting by Marker . . . . .	H-2
H-3 Product with DDP . . . . .	H-4
H-4 TLV Encoded Structure . . . . .	H-4
H-5 Taxonomy . . . . .	H-5
H-6 CCSDS I Class Unit Instance . . . . .	H-6
I-1 Structure of a Data Unit . . . . .	I-2
I-2 Object Access Software Layers . . . . .	I-7
J-1 Forms of Record Types . . . . .	J-1
K-1 Diagram of an "Image Cube" . . . . .	K-10
K-2 Diagram of Processing History for a File . . . . .	K-17
K-3 Textual Description of Processing History for a File . . . . .	K-18

## Chapter 1

### INTRODUCTION

This document is intended to serve the community of scientists and engineers responsible for preparing planetary science data sets for submission to the Planetary Data System. These sets include restored data from the era prior to PDS, mission data from active and future planetary missions, and data from earth-based sites. The audience includes personnel at PDS Discipline and Data Nodes, mission science investigators, and ground data system engineers including SFOC (Space Flight Operations Center) engineers.

In order for a data set to be used by those not involved with its creation, certain supporting information must be available. That information enables effective data set access and interpretation. Therefore, standards of quality and completeness have been developed that are to be addressed before PDS will accept a data set for distribution to the science community.

The interchange of data is increasingly important in planetary science. Electronic communication mechanisms have grown in sophistication. Data transfer that would have occurred by mailing tapes several years ago is now handled routinely over the Space Physics Analysis Network (SPAN). PDS will support the interpretation of a wide variety of transferred data by users with varying degrees of experience and available resources. Also, the use of new media for data storage and transfer such as CDROMs requires format standards to ensure readability and useability. The PDS has therefore developed a nomenclature that is consistent across discipline boundaries and standards for labeling data files. The 1986 PDS Interactive Data Interchange (IDI) workshop and the resulting compact disk product proved that a collection of science data from disparate disciplines can, once described in a uniform manner, be made readily accessible to a large group of users.

The current standards are presented here. Minor changes to this standard are expected, and this document shall be updated to reflect such evolution.

#### 1.1 DOCUMENT OVERVIEW

The overall organization of the document is shown in Figures 1-1 and 1-2. The document discusses three major topics related to data set preparation. The majority of the document is devoted to standards to be followed in submitting data. Another portion of the document describes the procedures to be followed in submitting data sets. The currently smallest portion of the document describes tools that are available to the planetary science community for use in the preparation or exchange of data sets. This last portion of the document will increase in size as more tools are developed by the Central Node and Discipline Nodes.

Chapter 2 outlines the process of preparing and submitting data sets, and gives contacts for further information. In Chapter 3 standards for documenting data sets and documentation examples are presented. Chapter 4 describes catalog standards employed within PDS, and points to the structural model used for the PDS Catalog. Nomenclature standards that are used within the catalog are given in Chapter 5, along with naming conventions for files and directories. Chapter 6 gives an overview of data format systems, and presents details of the Object Description Language (ODL) which is the system employed by PDS for generation of PDS labels. Miscellaneous standards important to the planetary science community are given in Chapter 7, including time, units, cartography, and the system for modular treatment of geometry data known as the SPICE concept. Chapter 8 addresses tools that are available or forthcoming from PDS. Information that is relevant but peripheral to these chapters or is more detailed has been placed in the appendices.

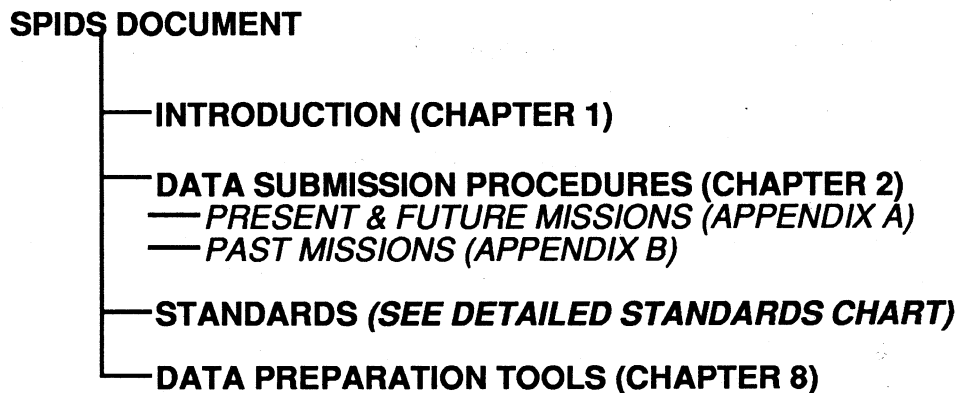


Figure 1-1: Major Components of the SPIDS Document

A companion document to this one, the *PDS Data Set Preparation Workbook*, is being written to provide step by step guidance in assembling materials for submission to PDS. It will contain procedures for the generation of PDS labels, sources for reference material, and information about how to organize both data and the catalog information to accompany the data.

## 1.2 ACKNOWLEDGEMENTS

This work is the result of many individual efforts over the last five years. The authors would like to acknowledge the contributions of PDS Central and Integrated Science Testbed Node staff in general, and in particular the work of Chuck Acton, Fred Billingsley, Randy Davis, Elaine Dobinson, Eric Eliason, Ed Greenberg, Bill Henslin, John Johnson, Hugh Kieffer, Bob Mehlman, and Larry Soderblom. Valuable comments were incorporated from Ray Arvidson, Dave Childs, Tom Duxbury, Peter Ford, Greg Kazz, Bill Kurth, Tom Renfrow, Dick Simpson, and Jim Torson.

## 1.3 REFERENCES

The following reference sources are mentioned in this document:

Batson, R. M., (1987) *Digital Cartography of the Planets: New Methods, its Status and Future. Photogrammetric Engineering & Remote Sensing* **53**, 1211-1218.

Consultative Committee for Space Data Systems (CCSDS) "Blue Book"; CCSDS 301.0-B-1; January 1987.

Davies, M.E., *et al* (1986) Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1985 *Celestial Mechanics* **39**, 103-113.

*General Data Interchange Language*, JPL Document D-3606, F. Billingsley, January 12, 1988  
*Guide on Data Entity Naming Conventions*; NBS Special Publication 500-149.

*Planetary Data System Data Dictionary*; JPL D-4854; January 15 1988.

*Planetary Data System Guidelines for Project Data Management Plans*; JPL D-5111; July 1 1988.

*Planetary Data System Software Management Plan*; JPL D-3487; May 3 1988.

*Planetary Data System Writing Conventions and Document Standards*; March 31, 1988.

*SFDU Usage and Description*; JPL D-5325; March 7 1988.

## STANDARDS

- **DOCUMENTATION (CHAPTER 3)**
  - CONTENTS (SECTION 3.4)
  - *FORMATS AND WRITING CONVENTIONS (APPENDIX C)*
  - *EXAMPLES (APPENDIX D)*
  
- **CATALOG (CHAPTER 4)**
  - ARCHITECTURE (SECTION 4.1)
  - CONTENTS (APPENDIX E)
  
- **NOMENCLATURE (CHAPTER 5)**
  - DATA NAME SYNTAX (SECTION 5.1)
  - FILE NAME SYNTAX (SECTION 5.2)
  - *DICTIONARY (APPENDIX F)*
  - *ABBREVIATIONS LIST (APPENDIX G)*
  
- **DATA FORMATS (CHAPTER 6)**
  - DATA REPRESENTATION (SECTION 6.1)
  - OBJECT DESCRIPTION LANGUAGE CONCEPTS (SECTION 6.2)
  - *SFDU CONCEPTS (APPENDIX H)*
  - *ODL SPECIFICATION (APPENDIX I)*
  - *DATA UNITS (APPENDIX J)*
  - *DATA OBJECTS (APPENDIX K)*
  - *DATA OBJECT SAMPLES (APPENDIX L)*
  
- **MISCELLANEOUS (CHAPTER 7)**
  - TIME (SECTION 7.1)
  - UNITS (SECTION 7.2)
  - BIN SIZE (SECTION 7.3)
  - SOFTWARE (SECTION 7.4)
  - GEOMETRY (SECTION 7.5)
  - CARTOGRAPHY (SECTION 7.6)
  - *SOFTWARE CODING (APPENDIX M)*

Figure 1-2: Standards in the SPIDS Document



## Chapter 2

### DATA SET SUBMISSION PROCESS

This chapter describes the process for submitting data sets to the Planetary Data System. It describes the various types of data suppliers and PDS support for the preparation and submission process, and presents a checklist of procedural steps to be followed in the submission process.

#### 2.1 DATA SUPPLIERS

Data sets are expected to enter the Planetary Data System in several ways: flight project data will be supplied according to agreements expressed in the Project Data Management Plans; some flight-relevant data will come directly from the Space Flight Operations Center (SFOC); restored and higher level data will be supplied by PDS Discipline and Data Nodes.

#### 2.2 SUPPORT FOR DATA SET PREPARATION

Data set preparation by a flight project is negotiated between project personnel and the PDS. These agreements are documented in the flight project's *Project Data Management Plan*, which are signed off by both project and PDS management (See Appendix A).

Restoration and preparation of existing flight and related data sets is undertaken principally by PDS Data Nodes associated with PDS Discipline Nodes. The data restoration is coordinated or overseen by the associated Discipline Node. Data Nodes exist as a data restoration entity only as long as necessary to transform the data set. Data Nodes function in cooperation with a Discipline Node on this temporary basis to treat a specific data set of interest to the community. Data Nodes are selected competitively on a regular basis. Refer to Appendix B.

#### 2.3 PDS CONTACT PERSONNEL

Planetary Data System staff are available to help interpret and implement the guidelines, recommendations, and standards contained in this document. Specific contacts include:

T.Z. Martin	Jet Propulsion Lab 4800 Oak Grove Dr. Pasadena, CA 91109 M/S 169/237	Tel: (818)354-2178 NASAMAIL: TZMARTIN SPAN: JPLPDS::TZMARTIN
M.D. Martin	Jet Propulsion Lab 4800 Oak Grove Dr. Pasadena, CA 91109 M/S 233/208	Tel: (818)354-8751 NASAMAIL: MIKEMARTIN SPAN: JPLPDS::MMARTIN
T. Duxbury	Jet Propulsion Lab 4800 Oak Grove Dr. Pasadena, CA 91109 M/S 183/501	Tel: (818)354-4301 NASAMAIL: TDUXBURY SPAN: NAIF::TDUXBURY
M. Johnson (Mission interface issues)	Jet Propulsion Lab 4800 Oak Grove Dr. Pasadena, CA 91109 M/S 301-320	Tel: (818)354-1493 NASAMAIL: MJOHNSON SPAN: JPLPDS::MJOHNSON

## **2.4 PROCEDURE FOR DATA SET SUBMISSION**

The process of submitting data to the PDS is similar in form whether done by a Discipline Node, a Data Node or a flight project, but the complexity and communication paths differ.

### **2.4.1 Estimation**

It is essential that the scope of the data set preparation task be defined at the outset, because resources are allotted at that time. Scope changes during execution of the task will be difficult to handle. This document is intended to assist in scoping data preparation tasks. The scoping process is done in consultation with PDS personnel, who will be able to supply the level of effort needed in previous interactions, and who will know how much detail is needed in a task.

### **2.4.2 Specification**

A task plan shall be prepared to describe the work to be done in data set preparation, including the tasks to be accomplished, the deliverables from the task, the schedule for this task, and the people responsible for the work and the deliverables. If the work is to be funded by the PDS then budget and other resource estimates must be included. The person responsible for the data preparation task needs to make sure that the following tasks are covered in the task plan:

- (1.) Documentation: writing text about the various aspects of the experiment (see Section 3.1 below). Clarity of writing is important. The PDS strongly encourages the writing of experiment and instrument papers and journal articles, and use of these to satisfy PDS requirements. Existing papers and articles should be used when available. Supplementary information may be needed when existing documentation is used, but does not provide all the needed information.
- (2.) Catalog data: preparation of information for the PDS Catalog. This material consists both of descriptive text and parameters, and will be used to give PDS user a view of the data set's many attributes and its applicability to his/her interests. In general, this catalog information is easy to prepare for those familiar with a data set. Nomenclature and definitions of these descriptive terms have been rigorously chosen and must be adhered to for uniformity across all disciplines and data sets.
- (3.) Software cleanup and coding: revision of existing software for clarity; new code to reformat data or adhere to standards. The data preparer should consider what software may be of benefit to other users, and avoid introduction of errors when revising software.
- (4.) Data reformatting: executing code to put the data into required formats, units, and nomenclature.
- (5.) Production: organizing the data onto the submission media.
- (6.) Quality assurance: checking what has been done prior to delivery.

The task leader should assign responsibility and provide resources for persons doing the tasks, and design schedules.

### **2.4.3 Preparation**

The following steps should be followed in the actual preparation of data sets:

- (1.) Determine primary data set content: a flight project will list its prime data sets in the Project Data Management Plan. See Appendix A. A Discipline or Data Node will follow the procedures described in Appendix B.



- (2.) Determine ancillary data and software content: given a selected primary data set what ancillary data and/or software must be included with the data set in order for it to be a complete archive product.
- (3.) Develop the appropriate SFDU structure for the data set: Appendix H of this document describes the SFDU structure. For more information consult the document "JPL SFDU Usage and Description (JPL D-5325)". The JPL Control Authority also provides consultation in the construction of SFDUs. New structures are registered with the JPL Control Authority.
- (4.) Develop the appropriate catalog data for each section of the data set: these data will be a K class SFDU. This catalog data will be recorded via the use of PDS Catalog Templates. These templates are machine readable forms developed by the data management staff of the PDS. The data preparer can use them to know exactly what he should supply for catalog data. Consult the PDS Data Dictionary for the appropriate catalog template for that data set. If a catalog template does not exist then consult with the Central Node to develop the template. Development of new items will follow the PDS nomenclature standards described in Chapter 5 of this document. The catalog templates provided by the Central Node will contain the catalog data elements required by PDS. The developer may wish to include additional items for clarity.
- (5.) Develop and describe the data format: Chapter 6 of this document describes the PDS data format standards.
- (6.) Document the data description of the data set: Chapter 3 of this document describes the PDS documentation standards.
- (7.) Build the data set: The structure of the registered SFDU will show how the various pieces of the data set fit together as a whole. The data set will be written on computer readable media.

#### 2.4.4 Submission

The prepared data set is sent to the PDS according to the agreed-upon schedule. Data sets are expected to be sent via standard media, such as:

- (1.) 1600 or 6250 bpi tapes
- (2.) transmitted over a network under an error-checking protocol
- (3.) optical disk: WORM or CDROM
- (4.) Typewritten documents and charts

PDS labels (see Chapter 6) are to precede the data in the files. These labels identify the data and its format, and simplify data cataloging. Properly constructed, the PDS labels provide much of the information required for the loading of PDS catalog entries. Catalog updating thus can be automated. The labels will also be used when data are delivered from PDS; they provide a means of ensuring that the data are readable, and in fact enable automatic recognition or ingestion of data.

The PDS recognizes that there may be instances where data are too voluminous to be reformatted to include the labels without substantial expense, or that such reformatting would require significant software modifications. The data preparer can negotiate to use detached labels in such situations. These are a separate set of label files which point to their associated data files (see Appendices I, J, K, and L).

The number of copies of the data required to be submitted to the PDS may depend upon the medium of choice, the volume, and the costs. In general, one copy should be supplied. The PDS will duplicate this as needed to support archive and distribution requirements.

#### **2.4.5 Review**

The submitted data set shall be received by the designated Discipline Node or the Central Node and checked for format and completeness. The data set must include the following: the science data, the catalog data, the documentation of the data set, any appropriate software and its documentation, any appropriate ancillary information and several examples of the data.

The Science Manager and Project Scientist shall select a peer review committee, and send the data set description and data examples to each member of the committee. The review committee shall include the Project Scientist or Science Manager or a substitute acceptable to both, a representative from the Central Node, the responsible Discipline Node manager and representatives from other Nodes who have some familiarity with the type of data under review.

The catalog data shall be verified by loading it into a development version of the Central Node catalog and shall be available to support queries for consideration during the peer review.

The review shall be conducted in a manner consistent with current peer review practices for consideration of science analysis proposals.

The review committee shall consider the data set format, content, documentation, ancillary data and software, and shall provide a written summary of all deliberations and conclusions. Each logical component of the data set shall be judged for inclusion or rejection, and if rejected, a clear indication of the reasons for rejection shall be provided. If a data set is rejected, the peer review committee will decide on a schedule for resubmission of the rejected component. Whenever possible, the explicit steps which must be taken by the supplier to correct faults shall be enumerated.

The review committee shall also make recommendations regarding the set of information which constitutes the minimum orderable data set, the assignment of curatorial responsibility, and the disposition of the data set. These recommendations shall include a determination of the appropriate data processing level and quality tags to be assigned, its position in the data hierarchy ( on- line, off- line, etc.) the number of inventory copies, and other recommendations to the Data Administrator.

Conflicts shall be resolved by mutual agreement of the PDS Project Manager, the Project Scientist, the Science Manager and the Discipline Node Manager. If the Discipline Node Manager is the data preparer, then conflicts will be resolved without his participation.

#### **2.4.6 Rework**

Discrepancies in the submitted material's quality, quantity, or continuity will be discussed with the supplier, and rework will be negotiated between the Discipline Node, PDS management, and supplier. It is hoped that frequent contact between the PDS representatives and suppliers will minimize the necessity of reworking submitted data.

#### **2.4.7 Signoff**

Satisfactory completion of a data preparation/submission task will be approved by the PDS management, reported to NASA management, and included in PDS publications.

## Chapter 3

### DOCUMENTATION STANDARDS

Accurate and complete documentation is required to make planetary data widely useable to the community; the absence of enforceable documentation standards has resulted in the existing level of difficulty in using many data sets. The documentation standards listed below are implemented by the PDS to ensure that all ingested data sets will be readily useable to the community at large.

Note that documentation may include descriptive information that can appear to the PDS user as text in the catalog or as help. Information supplied by the data preparer will either be destined for catalogs or for stand-alone material. A summary of major PDS catalog entries is provided in Appendix E.

#### 3.1 RECOMMENDED FORMAT

Documentation prepared for submission with data sets should follow the conventions described in *PDS Writing Conventions and Document Standards*, March 31, 1988.

The documentation section of the submitted material is to be in the form of text, with tables and figures supplied as needed to clarify the subject. Electronically based documentation is strongly preferred, as much of this information will eventually be archived in that form (perhaps on optical disks), and be made widely available. Hardcopy material may be readable in some cases by PDS scanners; if the data preparer plans to submit hardcopy material, he should coordinate this with the Central Node staff in order to maximize the possibility of PDS's being able to scan the hardcopy material.

Text files formatted in one of the major document processing protocols are strongly recommended. The formats which PDS can easily handle are Runoff, T<sub>E</sub>X, Word, Wordperfect, Wordstar and Vi. Use one of these formats if your text processing software has the option of producing them.

The data section of the submitted material is to be in the form of computer-readable files if at all possible. Formatting information is given in Chapter 6. Non-electronic material can be read in some cases by PDS scanners as noted above.

#### 3.2 EXAMPLES

Refer to Appendix D for examples of data set documentation. The Voyager PWS data were documented in the PDS label format as part of that team's submission of their data. Also included in that Appendix is text that was generated as part of a data set restoration effort by the Radiometry Node of the Pilot PDS. That information was intended for use in an online VAX help file. Please note that in this case only very limited information was available about some aspects of the experiment.

#### 3.3 SOFTWARE DOCUMENTATION

Software used to generate, process, and analyze the data set constitutes an important part of the submitted material. This part is frequently of interest to other investigators. The extent to which they use existing software depends very much upon the clarity and modularity of that software. PDS has developed software design and documentation standards intended to promote interchange of software tools. The data set preparer is strongly urged to follow these guidelines in

the development of new software, and to modify existing code comments where possible to meet the intent of the standard (see Appendix M).

Documentation such as requirements, design, and user manuals, where available, are of interest to those who might adopt the software or need to do similar development. Please preserve this information for any software that may be inherited, or which has affected data processing. Also note that software intended for eventual use at a Discipline Node for general PDS users must be accompanied by appropriate systems engineering documentation: requirements, design, and user manuals.

### 3.4 DOCUMENTATION CONTENTS

The following categories of information are required for each data set submitted to the PDS. Brief examples of the kind of information desired are given; note that these examples are illustrative and lack the amount of detail generally appropriate.

#### (1.) Instrument information

##### (a.) Measured parameters; theory of operation

What physical parameters are sought? What is the mechanism by which these measurements are obtained?

Example: "The single measured parameter of the UVEX instrument is ultraviolet radiance within the 300 to 400 nm band."

Example: "The instrument comprises a single optical window that is also the wavelength-defining filter, followed by a field-defining aperture, followed by a single bolometer detector chip. Signal output is related through calibration measurements of standard UV sources to UV radiance in the subject passband and limited by the field of view of the detector/aperture combination."

##### (b.) Sensitivity

What are the limits, in physical units, of measurable parameters?

Example: "The instrument response covers a usable range from  $3 \times 10E-11$  gauss, which is the noise level for the standard 1 sec integration, to  $8 \times 10E-5$  gauss, at which point a maximum data number (DN) level of 1024 is reached in the lowest gain state."

##### (c.) Temporal/spatial/spectral resolution

What are the inherent resolution limits of the experiment?

Example: "With the filter wheel fixed, samples may be obtained as frequently as 0.01 sec."

Example: "The optical field of view is circular with a diameter of 2.5 milliradians."

Example: "Spectral resolution is limited by the grating size to 0.6 nm within the 1200 - 1300 nm band, and to 1.0 nm in the 1300 - 1500 nm band."

##### (d.) Modes of operation; typical sequencing; observational compromises

What modes of operation are possible? What typical sequence of operations is performed during data gathering? What compromises limited effective usage of the instrument?

Example: "The instrument may cycle through various filter wheel positions (see table) or sample continuously in any chosen filter position. A typical sequence involves cycling through both radiometric and photometry filters with a period of 10 sec. The advantage of obtaining multiple wavelengths is offset by the time taken to cycle through filters, and

the consequently slower permissible scan rate for the platform. Other instruments on the scan platform prefer rapid scan rates, and so data collection was often performed with filter cycles of 2-3 sec, and just 3 or 4 filter positions employed in the cycle."

(e.) Calibration techniques; laboratory and in-flight behavior; techniques; problems

How was calibration performed before or during operation of the instrument? How were these measurements employed during the data reduction process?

Example: "Pre-flight calibration consisted of observations of blackbody targets in a thermal vacuum chamber for various instrument temperatures. The radiance versus measured DN profiles were used to develop lookup tables used in the reduction of flight data from DN to radiance."

(f.) Deviations from nominal performance

What unforeseen occurrences limited effective data gathering? What operations were taken to deal with these problems?

Example: "The instrument's drive motor developed erratic behavior after 3 years in orbit, such that the internal reference plate viewing mode could not always be exercised reliably. In response, the team chose to use commanded reference viewing rather than automatic cyclical viewing; the plate was viewed less frequently, and consequently data obtained after July 1 1993 have only 2% photometric accuracy versus 1% prior to that time."

(g.) Specifications

What are the basic instrumental characteristics?

(2.) Ancillary information

(a.) History of development

Trace the origin of the instrument or experiment and subsequent development that may be relevant to data interpretation.

(b.) Team personnel and current contacts

Who constituted the experiment team? Of those, who is currently a source of information? Who else has become a source?

(c.) Bibliography - instrument and science

What are the basic papers published by the team members relevant to this instrument? Where are the engineering specifications documented?

(d.) Data reduction software requirements; approach; implementation

What formal documentation of reduction software is available?

(e.) Data analysis software; tools for treating data of this type

What software exists to aid the user of this kind of data? Note: PDS encourages the submission of software tools along with data sets, provided these tools can be readily understood and portable.

(f.) Catalog information - existing catalogs used with the data set

Any set of information that would help a user browse, characterize, or otherwise subdivide the data set. Note that some of this information may be appropriately supplied in the PDS labels attached to the data.

(g.) Data format information

This material is normally contained in PDS labels associated with the data set itself, but it is valuable to have this information available in several forms.

The basic guideline for the content of documentation is, "Can this information be useful to a data user?" and "Is the material necessary, and is it sufficient?" There are many levels of inquiry possible regarding data sets, from the casual examination to the total reworking of a data set. What is useful therefore also varies. The intent of PDS is to err on the side of completeness; it is intended that calibration information, for example, be available to those who may want to reprocess data, or who question conclusions based on that calibration.

How much volume of information is expected? The answer depends on what is available, and on what is useful. If little is available, little is all that can be expected. If a great deal of information exists, then the answer is, "All that would be of interest and can be provided".

The data set supplier may not have access to all the information we would like about the experiment or data. Information and people may no longer be available; tapes may have been recycled. There may be insufficient time or funding to do a complete job. These are the realities of PDS as well as the rest of planetary science. Our goal is to do the best we can to serve the community within given constraints. The same goal should be kept in mind by the data set preparer.

## Chapter 4

### CATALOG STANDARDS

The PDS is developing a general purpose catalog (also known as a high level catalog) to encompass data sets from all disciplines of planetary science. As data come under the purview of PDS, appropriate catalog entries are made to recognize the existence of the new data. More detailed level catalogs give users insight into the nature of the data and provide information about time and spatial coverage or provide specialized browse data subsets.

One important goal of this SPIDS document is to describe standards used in the development of these catalogs. This is done to aid the development of catalogs by flight project staff, who have similar catalog needs during mission operation. Since PDS will acquire these catalogs from the projects, it is of mutual interest to promulgate a standard approach to database design.

The PDS has used relational database designs, and recommends the use of this database structure throughout the planetary community. Use of relational databases will simplify the transfer of information between flight projects, the science community, the PDS, and the NSSDC.

#### 4.1 PDS SCIENCE CATALOG ARCHITECTURE

The PDS science catalogs contain meta-data which characterize and describe the PDS science data holdings. This information is organized logically into two levels: a high level and a detailed level. This distinction is made for efficiency to avoid unnecessary duplication of information. The following discussion uses a bottom-up approach to characterize the contents of the two catalogs and distinguish them from the underlying data the catalogs describe.

##### (1.) *The Data Level*

The actual and processed measurements from the science and engineering instruments, as well as all of their derived data products, form the data level.

##### (2.) *The Detailed Level Catalog*

The next level up consists of detailed catalog information for the various disciplines. In a fully-configured PDS, there will be as many detailed-level catalogs as there are Discipline Nodes. Each catalog contains specific information about the data which is provided by that discipline. This discipline-specific information is needed both for correct interpretation of the data and for constraining searches.

##### (3.) *The High Level Catalog*

The high-level catalog contains generic information that cuts across all disciplines, and is therefore at a higher level of abstraction than the discipline-specific detailed-level catalogs. While the actual entries in the high-level catalog are discipline-specific, the information types are used to characterize all planetary data.

It is important to realize that the actual values stored in the two catalogs are specific to the data set and discipline being described, even if that information is part of the high-level catalog. It is the information type rather than the content that determines what catalog level is appropriate. The parameter measured may be specific to a single discipline (e.g., radiance or proton rate), but all disciplines measure parameters. Other information such as sampling intervals and the names of instrument subsystems also cross discipline boundaries, and are therefore present in the high-level catalog. Information is pushed down into the detailed-level catalog only when its type is specific

to a single discipline (such as the location of one reticle point of an image, or a range of brightness temperatures measured for an orbit sequence). Information which is even less general, such as the set of radiances which form an image, or the set of brightness temperatures measured in an orbit sequence, is considered to be data, and is not part of either catalog.

An example using the Fields and Particles discipline magnetic field data can serve to illustrate the three levels. The magnetometer subsystem on Voyager recorded magnetic field intensities in three orthogonal directions. Those intensities are the data. The data acquired can be analyzed in hourly increments, and information such as the percentage of data available during that hour and the type of data contamination which affected that hour's data can be derived. Such hourly details are not available and perhaps not important for other disciplines' data, and so the hourly details are a type of information which is stored in the Fields and Particles Detailed-level Catalog. The processing histories of the various magnetometer data sets are also unique to those data sets. However, the type of information which describes those processing histories, such as the names of the programs used and the hardware environment which ran them, is important to the processing histories of all data. Thus, the magnetic field intensities are data, the hourly contamination information is detailed-level catalog information, and the processing history description is high-level catalog information.

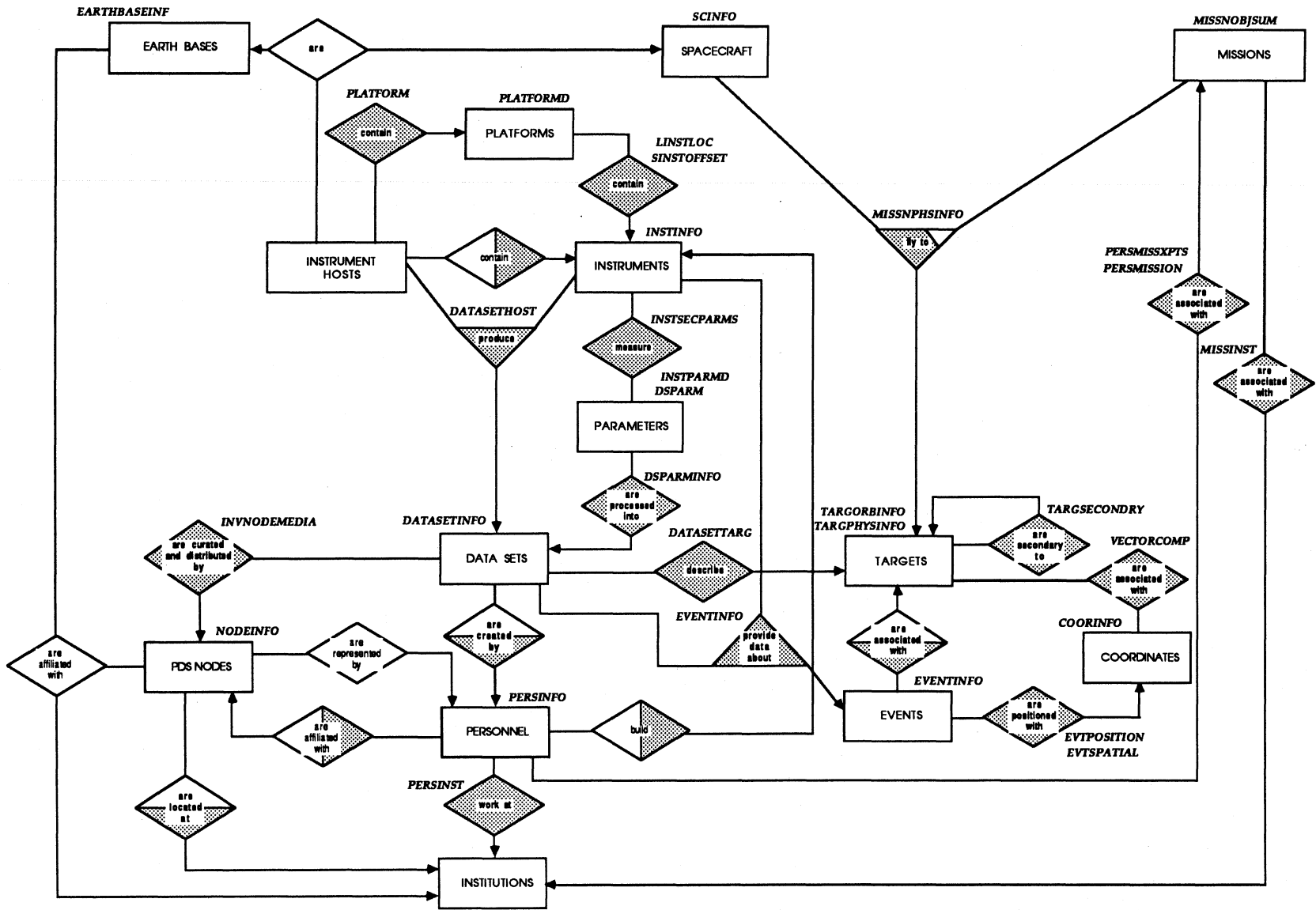
Note that the volume of information available decreases from the data level up through each catalog level, and that there is a corresponding increase in the generality of the type of information available.

## 4.2 ENTITY RELATIONSHIP MODEL

The definition and structure of the entities used in the PDS Catalog is shown in Appendix E of this document. Database "entities" are broad groupings of information, and are comprised of specific database "elements". A definition of elements contained in each entity is available in the *PDS Data Dictionary*, D-4854. Submitters of PDS data sets will be given templates to record the appropriate catalog data with their data sets. The templates indicate the criticality of a particular element in relation to a specific entity. An element that is marked critical must be included in the catalog data for acceptance of that data set by PDS. Figure 4-1 shows the entity relationship model for the high level catalog.



Figure 4-1: High-Level Catalog Schema





## Chapter 5

### PDS NOMENCLATURE STANDARDS

This Chapter describes the PDS nomenclature standards, which define the set of rules for constructing data object names. The purpose of establishing a standard syntax for data names is to facilitate user access to a system's data. It is particularly important to use common nomenclature in database management systems, where searches are made covering a variety of disciplines, techniques, and spacecraft missions. For a given data object, it is intended that any user of a system will be able to construct the same unique data object name by following these rules.

The traditional method of assigning names to data objects is best described as a process of eclectic nonchalance. Data producers and users construct names that are individually meaningful, perhaps following a personally preferred abbreviation scheme; the name chosen by different individuals is usually not the same. Witness the widespread use of SC, SPC, S/C, etc., all meaning SPACECRAFT. A solution was required for PDS that would allow for the consistent naming of data objects used in data dictionaries, catalogs, keywords, and documentation. The power of uniformity in naming cannot be over-stressed; it is fundamental to the comparison of data across discipline boundaries.

Several organizations have succeeded in developing a procedures for assigning standard names to data objects. The method adopted by the PDS is a derivative of the "OF language" developed by IBM. It also follows closely the publication *Guide on Data Entity Naming Conventions*, NBS Special Publication 500-149.

The objective of the PDS nomenclature standards is to create an environment wherein different individuals, working independently, will easily be able to construct the same name for a given data object. This objective, if achieved, would eliminate multiple names for the same object (synonyms), and duplicate names for different objects (homonyms); it would greatly simplify the task of browsing data dictionaries for those who are unfamiliar with its contents.

The construction rules must yield data object names which are easily grasped, are as consistent as possible with the common usage within the science community, and are also logically and methodically constructed, ideally from a predefined dictionary of component terms.

The standards described within this Chapter will be used in the PDS Catalog system and the PDS data file labels. The standards are also for use by submitters of data to the PDS in the creation of their mission catalogs and file labels.

The PDS strongly urges adherence to the standards set forth here within the planetary science community. Products submitted to PDS are required to use these standards. Note that the *PDS Data Dictionary* (a separate document, PDS D-4854) is intended to be a growing document; definition revisions will be admitted to make the data elements more widely usable. The PDS will endeavor to add any needed items which are not in the dictionary.

A substantial amount of effort by many parties has been invested in the development of these standards. It is hoped that the result is robust enough to be incorporated by other systems that deal with planetary data (e.g., NSSDC, SFOC, and flight projects).

#### 5.1 PDS DATA NOMENCLATURE SYNTAX

The PDS Data Dictionary contains the standard keyword names used to catalog PDS data products. An understanding of the syntax is necessary for two purposes: 1) as an aid in finding an

already existing keyword and 2) creating a new keyword for inclusion in the data dictionary.

### 5.1.1 Construction of Terms

All terms should be constructed from standard ASCII alphanumeric characters and the underscore character. No special characters (e.g., "&", "\*", etc.) are permitted. Appendix M contains the valid character set. The first character of the first term of a name must be alphabetic.

The PDS naming syntax is not case-sensitive. For example, all the following constructs represent the same object name:

- (1.) spacecraft\_event\_time
- (2.) SPACECRAFT\_EVENT\_TIME
- (3.) Spacecraft\_Event\_Time

Be aware, however, that the system employed in some implementations might be case-sensitive (e.g., some "C" compilers). Care must be taken in such instances to avoid creating separate data objects whose names differ only by the case (i.e., upper or lower) of one or more of the constituent characters.

### 5.1.2 Order of Terms within a Data Object Name

In general, the grammar of a data object designator (name) is hierarchical; the most specific term is placed first, the next most specific, etc., terminating with the least specific or most general. For example, consider a phrase such as "the time of an event on the spacecraft". Removing the articles and prepositions yields "time event spacecraft". The most general term here is "time", and therefore will be placed last in the hierarchy. Next, ask the question "time of what?" Obviously, the answer is "time of an event", which indicates that "event" is more specific than "time". The question "what kind of an event?" is answered by "spacecraft", the most specific term. Therefore, the data object name will be spacecraft\_event.time.

A data object name starts with the most specific term, followed by a connector, the next most specific (i.e., more general) term, and so on, terminating with the least specific (i.e., most general) term. The terms in the name are connected by an underscore (-) or a hyphen(-). The underscore is the preferred connector and should always be used except where it is not supported by hardware or software.

Words used in the nomenclature syntax are also categorized in three groups as SPECIFIERS, DESCRIPTORS or CLASS WORDS. The format of an object name is made up as follows:

object name := [SPECIFIER] [DESCRIPTOR] CLASS WORD

### 5.1.3 Specifiers

Specifiers are terms such as START, STOP, SPACECRAFT, INSTRUMENT, TARGET, etc. A specifier will generally be the first word of an object name.

### 5.1.4 Descriptor Words

The next term of a data object name should be chosen from a streamlined list of well-defined generic "descriptor words". Examples of descriptor words include angle, altitude, distance, location, radius and wavelength. This list is maintained by the PDS Data Administrator. See Appendix F for the current proposed descriptor word list.

### 5.1.5 Class Words

Class words usually comprise the rightmost word in a data object name. The class word identifies the basic "information type" of the data object, where information type includes both the data type (numeric, character, logical) and a size constraint.

The use of a limited set of class words will:

- (1.) Eliminate the need for users and data processing software to constantly access a data dictionary to parse, interpret, query or display values.
- (2.) Add a greater level of structure and consistency to our nomenclature.
- (3.) Constrain the selection and use of data values.
- (4.) Promote automated operations such as validity checking.
- (5.) Promote the development of intelligent software.

Class words include DATE, FLAG, ID, MASK, NAME, NUMBER, RATIO, TIME, and TYPE. The various class words are defined in Appendix F.

If no class word is used as the right-most word in an object name the term "value" is assumed to be the last term in a data object name. For example, one would construct MAXIMUM\_EMISSION\_ANGLE or SOLAR\_CONSTANT, as opposed to MAXIMUM\_EMISSION\_ANGLE\_VALUE and SOLAR\_CONSTANT\_VALUE.

When the class word "count" would be appropriate, the object name can be abbreviated by making the descriptor word a plural. The plural form implies "the number of something", for example, "the number of bytes in a record". The PDS nomenclature syntax advises appending an "s" to a descriptor word to indicate the inverse of "per each" or "number of".

For example:

Data Object	PDS name
number of bytes in record	record_bytes
number of records in file	file_records
number of label records in file	label_records
number of lines in image	lines
number of samples in line	line_samples
number of suffix bytes in line	line_suffix_bytes

### 5.1.6 Abbreviations

There are two aspects to abbreviations: the use of abbreviations in the formal "long" names assigned to data objects, and the construction of terse object names for use in processing environments where names are restricted to 7, 8, 10, 12, or some other number of characters.

#### 5.1.6.1 Abbreviation of PDS Formal Data Object Names

The maximum length of a PDS formal data object name is 30 characters. The limitation of names to 30 characters is needed because of the limitations of the software engineering tools used by PDS. There are instances, therefore, when it becomes necessary to abbreviate terms within a name in order to comply with this limit. The rules for abbreviations are:

- (1.) Abbreviate only if necessary to fit a name within the 30 character limit.

- (2.) When abbreviation is required, it should be performed from the right to the left (most general to most specific terms), and should stop as soon as the name-length restriction is met.
- (3.) The abbreviations for all dictionary terms are maintained by the PDS Data Administrator. There may be multiple allowable abbreviations for a number of terms. This is to support the construction of terse names of varying length (i.e., 12, 8, or maybe even 6 characters), while maintaining maximum readability. Each abbreviation, however, will be unique and correspond to one and only one full word. Appendix G contains the PDS Abbreviation List.

#### 5.1.6.2 The Construction of Terse Data Element Names

The terse name for a given data object is based upon the "formal" full name of the object. As previously noted, different applications may impose different length restrictions on terse names (for example, the Britton-Lee Intelligent Database Machine (IDM) restricts terse names to a maximum length of twelve characters). Several different sets of PDS terse names may thus be required.

A list of twelve-character terse names for the data objects in the PDS Data Dictionary and PDS Catalog design is maintained by the PDS Data Management Team along with the list of thirty-character full names for those data objects. This terse name list is intended as a reference for use by database implementors at the PDS Nodes and by other PDS developers.

For practical considerations, the PDS will not maintain multiple standardized lists of terse names. Rather, the PDS has adopted the following methodology for formation of terse names for PDS Version 1.0 applications:

- (1.) Terse names are constructed from full names. Where a full name is short enough to meet the applicable name length restriction, the full name is used as the terse name except where data independence considerations dictate a different approach. For example:

full name	-	LAST_NAME
12-character terse name	-	LASTNAME
8-character terse name	-	LASTNAME

(Note: Data independence considerations could motivate alternate choices for either or both of the terse names in this example.)

Where a full name is longer than the applicable name length restriction, the full name is shortened using a word-by-word abbreviating approach. For example:

full name	-	INSTRUMENT_NAME
12-character terse name	-	INSTNM (only 6 chars. needed)

- (2.) Abbreviations used in terse names are selected from the PDS Abbreviations List, which is maintained by the PDS Data Administrator. For some words, this list provides multiple allowable abbreviations to facilitate the construction of the shorter (e.g., eight-character) terse names.

In forming terse names for data objects which are included in the PDS Catalog schema - or for data objects which are to be added to the PDS Catalog schema - the need may arise for abbreviations which are not yet included in the PDS Abbreviations List. If this occurs, the PDS Data Administrator should be informed of the need. The Data Administrator will approve or suggest an alternate for the needed abbreviation, and will add approved new abbreviations to the Abbreviations List.

- (3.) In the process of word-by-word abbreviation of a name, the longest standard abbreviation for each word is used first. Once each of the words in a name has been abbreviated in this

manner, further abbreviation takes place using increasingly shorter standard abbreviations for each word until the applicable name length restriction is met. For example:

full name	-	INSTRUMENT_MODE_DESCRIPTION
12-character terse name	-	INSTMODEDESC
8-character terse name	-	INSMODDC (uses shorter abbreviations)

- (4.) As a rule, a terse name is formed without dropping any words from the given full name. In extreme cases, it may not be possible to form a meaningful terse name of the required length without dropping a word from the name. In these cases, the word dropped should be the word in the full name which is least essential to conveying the meaning of the data object.

For example, if it should become necessary to drop a word from the full name "COORDINATE\_SYSTEM\_REF\_EPOCH" in order to comply with a terse name length restriction, "REF," or REFERENCE, could be dropped. In this full name, REFERENCE is the word least essential to conveying the meaning of this data object. In this example, then, the terse name could be formed from abbreviations for the three remaining words, as follows:

full name	-	COORDINATE_SYSTEM_REF_EPOCH
8-character terse name	-	CRDSYSEP

- (5.) Word separators (for example, underscores) in terse names enhance the readability of the names but are costly in terms of character. Because of this overhead and the resultant loss of information content in a terse name, word separators have not been used in the twelve-character terse names in the design of the PDS Catalog IDM database.

## 5.2 PDS FILE NAME SYNTAX

In order to maintain compatibility with various computer architectures, file names should be chosen which utilize an eight character file name followed by a period and a three character extension.

### 5.2.1 Naming Rules

In cases where file names will contain an identification value constructed from the time tag or data object identifier, the following forms should be used:

Pnnnnnnn.EXT

where P is one of the following:

C = The following value is a clock count value (C3345678.IMG).

T = The following value is a time value (T870315.TAB)

I = The following value is an Image\_id (I242A03.IMG).

N = The following value is a numeric file identification number (N003.TAB).

#### 5.2.1.1 Standard File Extensions

The following file extensions shall be used wherever possible in the naming of Object Description Language data files.

Text files (Standard ASCII text)	filename.TXT
Table file	tablename.TAB
Image file	imagename.IMG
Cube file	cubename.CUB

Label file	objectname.LBL
Data file (without labels)	objectname.DAT
Formatted text files	filename.DOC
Compressed file	imagename.IMQ

The following file extensions shall be used for all Britton Lee IDM support files (ASCII) residing on the VAX system.

Create tables (sql)	tablename.CTB
Create indexes (sql)	tablename.CDX
Store commands (sql)	commandname.SQL
Views(sql)	viewname.VUE
Table data (ascii)	tablename.TDT
Table permissions (sql)	tablename.TPM
Store permissions (sql)	commandname.SPM
Text type tables (key,seqno,text)	tablename.TXT
IDMCOPY (by table)	tablename.D
IDMDUMP (database backup)	databasename.DBD
IDMFCOPY data file	tablename.FCD
IDMFCOPY format file	tablename.FMT
FREEFORM Screen	screenname.FFS
Documentation	precisename.DOC

### 5.3 DIRECTORY NAMING AND USAGE CONVENTIONS

The use of directories on random access media allows individual data files to be logically grouped for efficient location and retrieval. Well chosen directory structures lead the user naturally through a hierarchy of more and more specific directories until the appropriate one is found. On the other hand, a poor directory structure can make it nearly impossible to find a data file on its media volume. Examples of poor directory choices are the inclusion of too many files at a single directory level (this makes it hard to find the target file, and can actually cause severe performance penalties in the case of CDROM media); or a selection where the directory key is not useful, such as the use of day-of month for example, resulting in data for different months being in the same directory.

High level directories which deal with data sets which cover the range of planetary science disciplines shall follow the standard NSSDC hierarchy for discipline and sub discipline organization. For planetary science this hierarchy is as follows:

Planetary Science. (Directory name "PLANET")

Planetary Body (Directory name = Mercury, Moon, Mars, Venus, Comet).

Sub-discipline (Atmosphere, Ionosphere, Magnetosphere, Ring, Surface, Satellite (use satellite name instead if numerous files exist).

Directories should be constructed to provide access to a "screenful" of file entries. Within large collections of similarly named files the groupings should be chosen to provide from 20 to 100 files per directory. Directory names will be assigned using the portion of the filename which encompasses all files in the directory, with "X's" used to indicate the range of values of actual filenames in the directory. As an example the directories for the Uranus Imaging CDROM disk will be as follows:

[ARIEL]	- 40 files
[MIRANDA]	- 39 files
[OBERON]	- 18 files



[TITANIA]	-	42 files
[UMBRIEL]	-	31 files
[UNKNOWN]	-	4 files
[URANUS.C2674XXX]	-	4 files
[URANUS.C2675XXX]	-	6 files
[URANUS.C2676XXX]	-	3 files
[URANUS.C2677XXX]	-	31 files
[URANUS.C2678XXX]	-	57 files
[URANUS.C2679XXX]	-	65 files
etc...		
[U_RINGS.C2675XXX]	-	82 files
[U_RINGS.C2676XXX]	-	49 files
[U_RINGS.C2678XXX]	-	3 files
[U_RINGS.C2679XXX]	-	10 files
etc...		
Total		800 files

The Uranus and U\_RINGS target bodies are further subdivided into subdirectories containing specific groups of spacecraft\_clock\_count values. For example, the directory [URANUS.C2674XXX] contains image file names ranging from C2674702.IMG to C2674959.IMG.



## Chapter 6

### DATA FORMAT STANDARDS

This Chapter provides an overview of data format standards and describes the PDS Object Description Language, a label system to be used to describe the format, contents and relationships between PDS data units. Throughout this discussion, the term "data unit" is used to represent what is commonly termed a "file".

#### 6.1 DATA FORMAT REPRESENTATION SYSTEMS

Numerous data representation systems are in use or in development within the NASA science community. There is currently a major effort to develop the Standard Format Data Unit (SFDU) as an umbrella for registering and describing these formats. The SFDU architecture is described in Appendix H.

A more specific set of descriptive information, generally applicable to a single data unit (file), is found in the user label or header label formats employed in many applications. Examples of such formats are the ANSI labelled tape, the Landsat tape format, FITS, VICAR2, FLATDBMS, and the Common Data Format (CDF).

These formats can be differentiated as either predefined structures where a template is used to extract values from their assigned position, or keyword structures where a keyword or tag is used to identify the information about the data unit.

##### 6.1.1 Predefined Structures

The ANSI standard label for magnetic tape is an example of a predefined label format. Also, most of the telemetry record formats used by JPL projects utilize predefined header formats. The predefined formats do not have to be fixed. They may actually consist of numerous templates, any one of which might be needed depending on some key found in an earlier portion of the header.

These architectures are very efficient in certain processing environments, especially where data volume is critical. They are less attractive in open environments for several reasons. First, they are inflexible and rely on the assumption that the user is able to determine in advance everything that will ever be needed in the label. Second, they require external definition, either hard-coded within a processing program, stored in a dictionary or written down on paper. Third, they can be difficult to transport or interpret. In many cases the parameters are stored in machine specific binary format and may be extremely difficult to extract, even given a template describing the format and contents.

##### 6.1.2 Keyword Structures

Keyword label systems present the descriptive information in text format. The Flexible Image Transport System (FITS) and Video Image Communication and Retrieval (VICAR2) label systems are used by the astronomy and planetary imaging communities to provide descriptive information to accompany files of digital data. Both systems are widely used in image processing environments.

A major advantage of these systems is that the descriptive parameters and values are carried with the data and can generally be typed out for inspection without sophisticated processing software. In addition, they are flexible, and new keywords can be added to meet changing requirements. Since the keywords and values are in ASCII text format, they can be easily processed on most computer hardware with any computer language.

## 6.2 PDS OBJECT DESCRIPTION LANGUAGE (ODL)

The PDS Object Description Language (ODL) is a keyword structure derived from the FITS and VICAR2 formats. It differs from these systems in allowing longer keyword names, extended value types, and by separating labels with carriage control to make them easier to create or display.

### 6.2.1 ODL Objectives

The top level objective of the ODL development effort is to promote the use of planetary data by a wide science community.

The ODL system is intended to provide a language to communicate data files between the PDS data management system and the users. This language will be used to present information about both the format and contents of a particular data file, and should also provide the capability to describe its relationships to other data files.

Another important function of the ODL is to enable the automatic extraction of catalog elements from the labels during the submission process to the PDS. Given appropriate labelling and software, it should be possible to update catalog information pertaining to a data set's size, location, version, parameters, and help text.

### 6.2.2 ODL Concept

Space science data are almost always transported, stored and manipulated with the aid of computers. Because we use computers to store and retrieve space science data, we often tend to think of our data as files and records within the computer. But there are several reasons why the file/record model of data is not a good model for us to use:

- (1.) Files and records are what the computer manipulates, not what a human scientist wants to manipulate. The scientist wants to think in terms of images, spectra, maps, etc., but since data are arranged as files and records the scientist must make a translation – either mentally or through the use of software – between images, spectra, etc. and files/records. In the best of all worlds the scientist could deal directly with images and spectra without having to know much about how they are represented within the computer. (Let's call this issue "Level of Abstraction". We will give each issue below a name in parentheses so that we can refer to it later).
- (2.) Files and records are (by design) very general concepts and in most computers records and files can hold almost any type of data conceivable. If you receive a file of data without being told what is in the file you will probably have a difficult time determining what the file contains. To determine what is in a file you need two types of information that are often not supplied with the file:
  - (a.) The format of data within the file: Are the data arranged as an array of binary numbers, ASCII text strings, or what?
  - (b.) The content of the file: What do these numbers or strings represent? Are they an image, a map or what?
- (3.) An individual who creates a file determines the content and format of that file, and it is not uncommon to find the same type of data (say an image) represented in different formats by different users. (*Format and Content*).
- (4.) Any relationships between data in one file and data in other files are difficult to determine. For instance, how do you know when two different types of files contain images that are compatible (i.e., that can be compared one to the other)? There is often no way to tell just from looking

at the files; you need some sort of external information to tell you if the image files have the same format and a comparable content. (*Data Relationships*)

- (5.) File formats, record formats, even the format of individual data items (e.g., floating point numbers) differ between brand X computer and brand Y computer, making it difficult to transport data files from one machine to another. (*Portability*)

An alternative to the file/record model of data is the concept of the data base. The principal data base model these days is the relational model and some commercial relational DBMSs are in use in space science data applications. However many types of space science data including images, spectra and maps cannot be effectively stored within or retrieved from commercial relational DBMSs because they are designed to handle only simpler data types like integer scalar numbers and text strings.

### 6.2.3 Data Object Description

Objects are entities that can be perceived and examined by man or machine. Space scientists examine many objects including planets, atmospheres, rings, moons, magnetic fields, etc. Often we cannot see or feel these objects directly; instead we rely upon other objects like spacecraft and scientific instruments to provide us with data. The instrument data are actually objects also, objects that can be subjected to scientific analysis like images, spectra, time-series and tables. An instrument data stream is often jumbled up with other instrument data streams and with spacecraft status streams during transmission from the spacecraft to the ground, but typically one of the first things scientists do with data streams from their instruments when they receive them is to reconstitute the data objects (the images, spectra, etc.) that the instrument actually collected. Thereafter most analysis is performed on these data objects, and the output of analysis programs is more data objects (calibrated images, maps, etc.).

A note of caution is necessary here: we use the term data object to mean two different things. The word can mean:

- (1.) The class of an object: Image, spectra, etc. The class of an object identifies key aspects of the object that determine how the data object will be interpreted and analyzed. "Object class" is to objects what "data type" is to low-level data in a programming language like FORTRAN (FORTRAN data types include integer, real, complex, and character).
- (2.) An instance of an object: an actual data object that can be manipulated and analyzed. An example would be the Voyager 1 wide-angle camera image shuttered at time T (there is only one such image). Each instance of an object belongs to one and only one object class; in our example, the camera image might belong to a class named RAW VOYAGER IMAGE. There may be hundreds or thousands of instances of a particular class of object.

Generally we will use the term "object" without specifying whether we mean class or instance, but the context should make it clear which we mean.

### 6.2.4 Data Objects Storage and Transportation

As with any type of computer data, a data object is nothing more than a collection of bits. Typically those bits are arranged into more meaningful types of data like numbers, text strings, etc, and those numbers and text strings are arranged into data objects like images, spectra, maps, etc. When we want to store or transport data objects we place them into "data units". Data units are essentially containers for data objects. A data unit may contain a single object, several objects of the same class or several objects of different classes. An example of several objects of the same class would be a data unit containing a number of Voyager images. An example of several objects

from different classes would be a data unit containing a Voyager image plus a histogram of that image (the image and the histogram are two different classes of objects).

We often implement data units as files but a data unit is more than a file because it must contain or point to information that describes the object(s) within the data unit. A data unit provides, either directly or indirectly, documentation on each of its objects, both in human terms and computer terms. By "directly", we mean that the data unit may contain a "label" that contains this documentation. By "indirectly" we mean that the data unit may contain only a reference to a database containing such documentation. The decision as to whether a data unit contains a full label or only a pointer to the necessary documentation is left to the person generating the data unit (who must weigh issues like the overhead involved, how recipients will use the data unit, etc.).

Each object within a data unit requires documentation both at the class level and at the instance level. The following information is needed to describe a class of objects:

- (1.) A name that uniquely identifies the object class and the relationship between this object class and other objects classes. The methods of constructing object class names and specifying class relationships are discussed later in this paper.
- (2.) A description of the format of the data object in terms that a human can understand and utilize. Taking a Voyager camera image as an example, the format information we need is that each such image consists of 800 scan lines with 800 samples per scan line and each sample is assigned an integral gray-level value in the range 0 - 255. All of this information is determined by the specific characteristics of the Voyager cameras, and none of this information specifies the way in which we will store the image within the computer: a class description is always independent of the way in which the objects within that class will be implemented.

For each instance of an object it is also useful, but not mandatory, to have the data unit include information that describes the content of the object.

The content of a data object is a function of the instrument object that created the data object and the real-world object which the instrument was examining, so we need information that ties data object, instrument object, and real-world object together. This holds equally true for data objects that are output by a data analysis program rather than directly by an instrument: data objects that are created by programs should be annotated with a history of the processes to which the data object has been subjected.

One of the chief ways of specifying data content is through time stamping, and most descriptions of data object content will contain the time at which the data object was created. To illustrate further the concept of identifying data content, let's continue the example of a Voyager camera image: useful information on content would include the identity of the target at which the instrument was pointed when the data object was created (Jupiter, Titan, Uranus, etc) and a synopsis of the instrument state at the time the picture was shuttered (the filter through which the image was taken, gain state, etc.).

Along with the information describing the format and content of each data object we need information that describes how the data objects are represented within the data unit. Since data units are manipulated directly by computers, this description needs to be in terms that computers can understand and utilize. Thus computers retrieving or receiving a data unit will know how the originator arranged the objects and the bits within the data units. This information must include:

- (1.) The location of each object within the data unit. This can be achieved through the use of pointers that point to the beginning of each data object within the unit, or similar mechanisms.
- (2.) A description of how the data object is broken into records and into underlying data types

like integers, characters, etc. Continuing with our Voyager camera image as an example, if we receive a data unit containing such an image we need to know that each scan line is contained within a 800-byte fixed length record and that each sample is encoded as an 8-bit unsigned integer quantity.

- (3.) If an object contains data types like integers, reals, and text strings that are machine-dependent, we need to know the conventions used for encoding such values on the originating computer so that if the receiving computer has different formats for those data types it can recognize the incompatibility and hopefully translate the received values into its own corresponding data types.

To achieve the full potential of the object-orientation we need a standard way of describing the format and content of data objects and data units. Since part of the information about form and content is designed for use by humans while the other part is designed for use by computers, it is useful to devise a language that is readable and writable by both humans and computers. We call such a language an Object Description Language. The PDS has developed an Object Description Language (ODL) which is described in detail in Appendix I.

### 6.2.5 Object Class Hierarchies and Inheritance

As mentioned previously, each instance of a data object belongs to one and only one object class. One of the most powerful concepts of an object-oriented approach is that object classes can be arranged into a hierarchy with classes in the lower levels of the hierarchy inheriting properties from their ancestors higher up in the hierarchy. This is called "class inheritance" and to see how it works let us develop a rough cut of the first level of a hierarchy for space science data objects:

IMAGE  
SPECTRUM  
TIME\_SERIES  
TABLE  
HISTOGRAM  
MAP

Descriptions of the formats of objects in the above classes can be expressed in an ODL, but the descriptions are necessarily vague: we cannot specify that every image will have X scan lines and Y samples per line because different cameras produce images with different values for X and Y. We solve this problem by providing only a template of the object format description for these high-level object classes. For example, a description of the class IMAGE might contain something like the following (given below in a pseudo-ODL):

LINES                   = positive integer  
LINE\_SAMPLES           = positive integer

·  
·  
·

This template indicates that for every image object the number of lines per image and the number of samples per line must be included in the format description. The text to the right of "=" is to be replaced with a value of the appropriate type.

We can now use this template to create new classes that are "subclasses" of class IMAGE. By "subclass" we mean that the class shares the characteristics of other images but differs in salient ways, namely that it has specific values for the variables in the template that may be different from the values for other classes. Using class IMAGE as an example: the cameras aboard different

spacecraft produce images of different sizes so that we need to define one subclass for each type of spacecraft. The object class hierarchy for images might then look like this:

- IMAGE
  - MARINER\_9 IMAGE
  - VIKING IMAGE
  - VOYAGER IMAGE
  - etc.

We can tailor subclasses in two ways: we can fill in template values for attributes that were inherited, or we can add new attributes that weren't inherited. We cannot, however, ignore attributes that are inherited. The format description of class VOYAGER IMAGE might include something like the following, again specified in our pseudo-ODL:

```

LINES                = 800
LINE_SAMPLES         = 800
SAMPLE_BITS          = 8
.
.
.
```

The first two pseudo-ODL lines above fill in template values inherited from class IMAGE. The third line adds a new attribute that is specific to class VOYAGER IMAGE.

The hierarchy can be extended as far as it is useful to take it. For example, when data objects are analyzed the processing programs used typically produce new data objects that are related to, but different from, the input objects. This leads us to extensions in the hierarchy like the following:

- VOYAGER IMAGE
  - RAW VOYAGER IMAGE
  - DESPIKED VOYAGER IMAGE
  - CALIBRATED VOYAGER IMAGE
  - etc.

The name of a class should reflect the class's place in the hierarchy. For example, the class name CALIBRATED VOYAGER IMAGE indicates the complete inheritance chain of calibrated Voyager image data objects.

### 6.2.6 Summary

The object-oriented data model uses the data object as the principal thing that is being stored, transported and manipulated (as opposed to files and records). Data objects are encapsulated in data units whenever they are stored or moved. Data units contain or point to descriptive information on the following:

- (1.) The data format of the object, expressed in terms that a scientist can understand
- (2.) The data content of the object
- (3.) The location and format of each data object within the data unit, in terms that a computer can understand

Therefore, the object-oriented approach resolves the following issues concerning the file/record model of data raised earlier:

- (1.) *Level of Abstraction:* Data objects represent data at the level at which scientists work - images, spectra, etc. - as opposed to the level at which computers work (i.e., files and records). It is more natural for the scientist to manipulate objects than files and records.



- (2.) *Format and Content:* Data objects are encased in data units and data units provide, directly or indirectly, a description of the format of the data object – both in human terms and in computer terms – plus a description of the content of the object. Each data object belongs to a class and the format, as described in human terms, is the same for all objects within a class, thus classes provide a means for achieving uniformity since everyone who generates an object of a particular class will have to format and describe that object in the same way.
- (3.) *Data Relationships:* Relationships between data objects are expressed through class hierarchies and inheritance. A class inherits the attributes of all its ancestors in the class hierarchy. Therefore it is easy to tell that an object of class MERCATOR MAP is related to an object of class SINUSOIDAL MAP because they are both subclasses of class MAP. It is also easy to tell that there is no direct relationship between a MERCATOR MAP object and a UV SPECTRUM object since they do not share ancestors in the hierarchy. Class format descriptions written in an ODL can be used to determine whether or not two classes of objects are compatible (they may or may not be compatible even when they share the same ancestors).
- (4.) *Portability:* Objects and data units are more portable than files because of the descriptive information that is contained in or that is pointed to by the data unit. A receiving computer can look at the format descriptions for the objects within a data unit and determine whether or not the objects are in a format that are compatible with that computer. If they are not, and there is appropriate software available, the objects can be transformed into the proper data format by the receiving machine.



## Chapter 7

### MISCELLANEOUS STANDARDS

#### 7.1 TIME

The representation of time within a database is of particular concern, since time is often used to constrain searches. PDS has adopted the ISO 8601 standard entitled "Data element and interchange formats – Representations of dates and times" for this purpose, and applies it across all disciplines in order to give the catalog generality. The standard is UTC (Universal Time Coordinated).

The 8601 standard covers the representation of the following:

- (1.) Dates consisting of year, month and day-of-month
- (2.) Dates consisting of year and day-of-year
- (3.) Dates consisting of year, week-of-year and day-of-week
- (4.) Clock times consisting of hours, minutes and seconds, including local time, UTC and alternate time zones
- (5.) Periods of times

This proposal calls for adoption of a subset of the representations allowed by the 8601 standard. It is important to note that the 8601 standard covers only ASCII representations of dates and times. For binary representations of dates/times we propose that the PDS adopt the time code format standards recommended by the Consultative Committee for Space Data Systems (CCSDS) in document CCSDS 301.0-B-1 (Blue Book), January 1987.

##### 7.1.1 Representations of Dates

Dates shall be represented as either year, month and day-of-month or as year and day-of-year using the full 8601 format, which has the fields separated by dash characters.

###### *Year, Month and Day of Month:*

Complete year, month and day-of-month: *ccyy-mm-dd*. 8601-compliant representations of dates as numbers only (*ccyymmdd*, etc) are not allowed under this proposal. The 8601 standard requires all digits of a field to be specified, using leading zeros as needed. For example the following is not a legal format for August 1, 1988: *1988-8-1*; the proper representation is *1988-08-01*.

Year, month and day-of-month for dates in the current century: *yy-mm-dd*. 8601-compliant representations where year and/or month fields are omitted (for example the use of *-dd* to represent a date within the current year and month) are not allowed under this proposal.

###### *Year and Day of Year:*

Complete year and day of year: *ccyy-ddd*. The 8601 standard requires all digits of a field to be specified, using leading zeros as needed. For example the following is not a legal format for February 1, 1988: *1988-32*; the proper representation is *1988-032*.

Year and day of year for dates in the current century: *yy-ddd*. We strongly recommend specifying the full four-digit year rather than the two-digit year-of-century.

### 7.1.2 Representations of Times

Times shall be represented as hour, minute and seconds using the full 8601 format. The hour, minutes and seconds consist of three two-digit fields separated by colons and modulo 24, 60 and 60, respectively. The seconds field may optionally have a fractional part; if a fractional part to seconds is specified, a period shall be used as the decimal point and not the European-style comma.

Local Time: *hh:mm:ss.s*

UTC Times: *hh:mm:ss.sZ*

Alternate Time Zone (Relative to UTC): *hh:mm:ss.s+n*

where n is the number of hours from UTC.

### 7.1.3 Dates and Times

Dates and times shall consist of any legal representation of date and any legal representation of time separated by the letter T. For example:

<i>ccyy-mm-ddThh:mm:ss.s</i>	represents a date and local time
<i>ccyy-mm-ddThh:mm:ssZ</i>	represents a date/time in UTC
<i>yy-dddThh:mm:ss+7</i>	represents a date and time in Pacific Daylight Time

### 7.1.4 Periods of Time

(To Be Supplied)

## 7.2 UNITS

The uniform usage of units is essential in a broadly-based catalog system, for obvious reasons. One cannot search for all the instruments covering 400 to 700 nm wavelength if some of the entries are in Angstroms and some in microns. The PDS Data Dictionary Report (a separate document, PDS D-4854) will define desired units for each database element used in the system. The standard is SI, Systeme Internationale. Therefore note that micrometers are preferred over microns, for example. There are a few exceptions to SI units to allow for consistency with the community standard usage. These exceptions are documented in the data dictionary.

The following summary of SI unit information is extracted from the *Chicago Manual of Style*.

*Base units* – As the system is currently used, there are seven fundamental SI units, termed “base units”:

	UNIT	ABBREVIATION
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

SI units are all written in lowercase style; abbreviations are also lowercase except for those derived from proper names. No periods are used with any of the abbreviations in the international system.

*Derived units* – In addition to the base units of the system, a host of derived units, which stem from the base units, are also employed. One class of these is formed by adding a prefix, representing a power of ten, to the base unit. For example, a kilometer is equal to 1,000 meters, and a millisecond is .001 (that is, 1/1,000) second. The prefixes in current use are as follows:

#### SI PREFIXES

<i>Factor</i>	<i>Prefix</i>	<i>Symbol</i>	<i>Factor</i>	<i>Prefix</i>	<i>Symbol</i>
$10^{18}$	exa	E	$10^{-1}$	deci	d
$10^{15}$	peta	P	$10^{-2}$	centi	c
$10^{12}$	tera	T	$10^{-3}$	milli	m
$10^9$	giga	G	$10^{-6}$	micro	$\mu$
$10^6$	mega	M	$10^{-9}$	nano	n
$10^3$	kilo	k	$10^{-12}$	pico	p
$10^2$	hecto	h	$10^{-15}$	femto	f
$10^1$	deka	da	$10^{-18}$	atto	a

Although, for historical reasons, the kilogram rather than the gram was chosen as the base unit, prefixes are applied to the term gram instead of the official base unit: megagram (Mg), milligram (mg), nanogram (ng), etc.

Another class of derived units consists of powers of base units and of base units in algebraic relationships. Some of the more familiar of these are the following:

	UNIT	SYMBOL
area	square meter	$m^2$
volume	cubic meter	$m^3$
velocity	meter per second	$m/s$
acceleration	meter per second squared	$m/s^2$
density	kilogram per cubic meter	$kg/m^3$
luminescence	candela per square meter	$cd/m^2$

Many derived SI units have names of their own:

	UNIT	SYMBOL	EQUIVALENT
frequency	hertz	Hz	cycles per second
force	newton	N	kilogram-meters per second squared
pressure	pascal	P	newtons per square meter
energy	joule	J	kilogram-meter
power	watt	W	joules per second
quantity of electricity	coulomb	C	ampere-second
electric potential	volt	V	watts per ampere
capacitance	farad	F	coulombs per volt
electrical resistance	ohm	-	volts per ampere

*Use of figures with SI units* - In the international system it is considered preferable to use only numbers between 0.1 and 1,000 in expressing the quantity of any SI unit. Thus the quantity 12,000 meters is expressed 12 km, not 12,000 m. So too, 0.003 cubic centimeters is preferably written 3 mm<sup>3</sup>, not 0.003 cm<sup>3</sup>.

For the decimal point, the international system permits either a dot (the British and American practice) or a comma (the French practice). Note that the comma is not used in international work to mark off groups of three digits in large numbers: if such figures cannot be avoided, spaces are left between the groups of three--to the right of the decimal point as well as to the left:

31 000 000                      0.000 000 31

### 7.3 BINNING

Certain derived data sets are worth curating because of their high value and wide applicability. Examples of these are maps of basic physical parameters such as albedo, thermal inertia, elevation, etc. that have been derived from measurements. It has been demonstrated by both the Lunar and Mars Consortium efforts that a uniform choice of binning or sampling intervals is of crucial value for the intercomparison of such data sets, and in fact makes such comparison simple if the data sets are available in a common database. We recommend here that if such data sets are developed for planetary bodies the data be binned in a simple cylindrical coordinate system, and that the bin sizes be binary multiples of 1 degree. Thus bin sizes of 0.25, 0.5, 1.0, 2.0, 4.0 degrees would be desirable. High density data sets may be easily re-binned in this scheme to intercompare with lower resolution data. See also Section 7.6 for cartographic standards.

### 7.4 SOFTWARE DEVELOPMENT

The PDS has developed a set of coding standards, which are essential in the design of any large software-dependent system. These standards are detailed in the PDS Software Management Plan (D - 3487); parts of that document appear also here in Appendix M for reference; adoption of these standards in the development of new code will enhance the future usage of that material by anyone and will smooth its incorporation into the PDS if such software is of interest to data users. It is not expected that existing code accompanying data sets will in general be converted. In the case of data processing software of archival interest, changes to the actual code structure are in fact dangerous. Comments, however, should be added to existing code to improve readability. Several document reader software tools are being developed within PDS to extract comments from code. Refer to Appendix M also for standards regarding commenting.

Please note that software which is being generated to implement the basic operational functions of the PDS (such as data retrieval, data preparation, and data delivery) must follow very stringent development guidelines; these have been detailed in the PDS *Software Management Plan*.

## 7.5 ANCILLARY GEOMETRIC INFORMATION (SPICE FILES)

In the past, geometric information for spacecraft experiments has been supplied to the teams through a central project facility that runs the appropriate software. There is a trend towards decentralization of this function, for several reasons. The data set supplier associated with future missions will encounter this concept, and it will affect what they need to pass on to PDS. Therefore, an introduction is provided below to the revised approach to generation of geometric data.

SPICE is a formalism for treating geometric and other ancillary information pertinent to the understanding of science data returned from instruments on planetary spacecraft. Those familiar with ground data systems supporting past NASA planetary missions may envision the SPICE system as, in part, a functional replacement for the Supplementary Experiment Data Record (SEDR) component.

An objective of the implemented SPICE system is that it, with the addition of the downlink science instrument data from the spacecraft, will contain all of the information needed to recover the full scientific value of the returned science instrument data, and it will facilitate correlations of individual instrument data sets with data from other instruments on the same or other spacecraft.

The SPICE concept is being executed within the larger context of the Planetary Data System, as it is within this environment that the full benefit of these concepts may be obtained. In turn, substantial replication of the PDS system architecture, including SPICE, within flight project environments will further extend these benefits.

The primary SPICE data sets, called "kernels", are those which contain the fundamental and irreducible set of ancillary information. Kernels are composed of information which comes from the most knowledgeable sources of such information, which has anticipated use within the SPICE system, and which have been structured, formatted and cataloged to PDS standards.

The name SPICE is an acronym (coined by Hugh Kieffer of the USGS in Flagstaff) from which reference to the kernels is made as follows:

S - Spacecraft ephemeris, or more generally, location of the observer, as a function of time.

P - Planet, satellite, comet or asteroid ephemerides, or more generally, location of the target bodies being observed, as a function of time. The P kernel also logically includes principal physical, dynamical and cartographic "constants" related to the target bodies, such as spin axis orientation and size and shape specifications.

I - Instrument description kernel. Contains descriptive and operational data peculiar to a particular instrument. Some examples of information included in the I kernel are mounting alignments, internal timing characteristics, geometric and radiometric calibration data and descriptions of operating modes. Note that I kernels, like the "constants" part of the P kernel, contain data that are largely time invariant.

C - Pointing kernel. The name derives from historical use of the letter C to refer to a 3x3 matrix defining pointing of a spacecraft's scan platform in inertial coordinates. The C kernel contains time tagged pointing angles for the (or a) major spacecraft structure on which science instruments are mounted. Pointing data for a specific instrument are obtained by combining appropriate portions of an I kernel with the C kernel.

E - Events kernel. The foundation and principal contents are derived from the Integrated Sequence of Events (ISOE file) used to produce actual spacecraft commands. The E kernel is then augmented with real-time commands and "notebook entries" from scientists and those monitoring data system and spacecraft performance. The E kernel and portions of the I kernel are the only SPICE system elements focussed beyond purely geometric information.

The "SPICE" acronym might better have been named "SPICES" since there is one additional and equally important component: SOFTWARE. The SPICE kernels alone do provide the data needed to interpret science instrument data sets, traditionally called Experiment Data Records, or EDRs. Pragmatically, though, provision to the user community of only the SPICE kernels would be shortsighted and unacceptable. Instead, the SPICE formalism specifies that the user community also be provided with the software needed to both read the kernels and, subsequently, compute the principal observation geometry parameters and retrieve related ancillary data needed to help evaluate the EDRs.

With these components of SPICE now identified, a premise and a major operational axiom fundamental to SPICE should be stated.

*PREMISE: The proliferation of inexpensive, high speed micro and mini computers and advances in data distribution technology facilitate the distribution of computation functions where appropriate.*

*AXIOM: The ability to compute observation geometry parameters and to retrieve easily allied ancillary experiment data is considered at least equivalent to having such data precomputed for the end user.*

This leads to the last major design concept of SPICE, which is the transfer of the capability and responsibility for computation of derived ancillary information (SEDR parameters, and more) to the end user – the scientist or PI team or engineering team.

Briefly summarized, then, the SPICE System Specification provides for the production at JPL of the elemental data files called kernels, and a portable software library, called the "toolkit", containing subroutines to read those kernels and compute most of the observation geometry parameters needed to aid interpretation of science instrument data sets. The toolkit software, including test/demonstration programs, is distributed to PI teams, with assistance from JPL. The PI teams integrate this toolkit software with their own analysis programs, functionally replacing that element of their software which used to read a SEDR tape.

Once a mission is underway, as new kernels are produced they are cataloged on a computer system accessible by PI teams. The PI may "order" all or selected kernels, and use these at his home site to compute or extract the geometry and related ancillary information of interest. The user does this according to his own schedule, and for only the specific time spans of interest. The user can fold in his own related software or otherwise change what is now his own "SEDR factory." (Changing SPICE toolkit software is not recommended.)

Further and more comprehensive discussion of these ideas, and details of SPICE component specifications will be published in a SPICE Primer at a later date.

## 7.6 CARTOGRAPHIC DATA

The following cartographic data standards were developed through an iterative process involving both the NASA Planetary Cartography Working Group (PCWG) and the PDS. Members of the PCWG are also on the key IAU committees which set these same standards for international adoption; therefore, the PDS adopted cartographic standards are consistent with the IAU standards. The PDS, rather than making unilateral decisions on cartographic data standards, looks to the PCWG as the controlling body for these standards within NASA and the PDS. It is recognized that the IAU continually reviews its standards and may, at some time, make a change affecting the cartographic standards. If this happens, the PDS will work with the PCWG and decide its course of action at that time.

A driving force for standards is to enhance the exchange and correlation of data sets between the



same instrument at different times and between different instruments and missions. The adoption of these cartographic standards, as with other standards, is a compromise between technical purity, tradition, standard practices, previous standards and potential costs associated with changes. Cost impact was an important consideration for some planetary bodies. The IAU standard for planetographic longitude definition rather than planetocentric was adopted because of traditional use. Therefore, technical purity may be compromised by these standards; however, data exchange and correlation is still enhanced by the adoption of these standards.

For historical reasons, Mars and Earth are exceptions to many of these standards, as noted below. The giant, gaseous planets (Jupiter, Saturn, Uranus, and Neptune), having no visible solid surfaces, are not explicitly covered by these standards, but the question of planetocentric vs. planetographic latitude for these bodies may need to be addressed in the future.

### **7.6.1 Inertial Reference Frame/Time tag/Units**

The Earth Mean Equator and Equinox of Julian Date 2451545.0 (referred to as the "J2000" system) is the standard inertial reference frame. The Earth Mean Equator and Equinox of Besselian 1950 (JD 2433282.5) is to be supported because of the wealth of previous mission data referenced to this system. The transformations between the two systems are to be available. Time tagging of data using UTC in Year, Month, Day, Hour, Minute and decimal Seconds is the standard, with Julian Date being supported. SI metric units, including decimal degrees, are the standard.

### **7.6.2 Spin Axes and Prime Meridians**

The IAU-defined spin axes and prime meridians relative to the J2000 Inertial Reference System are the standard for planets, satellites and asteroids where these parameters are defined. For other planetary bodies, definitions of spin axes and prime meridians determined in the future should have the body-fixed axes aligned with the principal moments of inertia, with the North Pole defined as along the spin axis and above the Invariable Plane. Where insufficient observations exist for a body to determine the principal moments of inertia, coordinates of a surface feature will be specified and used to define the prime meridian. It is expected that some small, irregular bodies may have chaotic rotations and will need to be handled on a case-by-case basis.

### **7.6.3 Reference Coordinates**

The Cartographic latitude and longitude are the standard reference coordinates. These coordinates are for a vector from the body center-of-figure to the surface. Latitude is measured from -90 degrees at the South Pole to +90 degrees at the North Pole. Longitude has values from 0 to 360 degrees and is always positive. Longitude is measured from the prime meridian using the same convention as for the IAU Planetographic longitude.

Mars and Earth are exceptions; Planetographic latitude is the standard for these bodies. The long history associated with this definition and the volume of data and derived data products using this convention makes a change to Cartographic coordinates impractical.

The proposal to adopt the Cartographic coordinate definitions has been made to the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Potential Elements of the Planets and Satellites by IAU/PCWG members. The final resolution on adopting these coordinate definitions has not been decided at this time.

#### 7.6.4 Reference Surface

The Digital Terrain Model (DTM), giving body radius as a function of Cartographic latitude and longitude in a sinusoidal equal-area projection, is the standard. Mars is to be an exception where Planetographic latitude is to be used. Spheroids, ellipsoids and harmonic expansions giving analytic expressions for radius as a function of Cartographic coordinates are to be supported.

The Digital Image Model (DIM) giving body "brightness" in a specified spectral band or bands as a function of Cartographic latitude and longitude in a sinusoidal equal-area projection, and associated with the surface radius values in the DTM, is the standard. Mars is to be an exception where Planetographic latitude is to be used. DIMs registered to spheroids, ellipsoids and harmonic expansions are to be supported.

#### 7.6.5 Map Resolution

The spatial resolution of a map will use  $1 / 2^n$  degrees as the standard. The vertical resolution will use  $1 \times 10^m$  meters as the standard, with  $m$  and  $n$  chosen to preserve all the resolution inherent in the data.

#### 7.6.6 Documentation

The PCWG and PDS support the concept of adopting these cartographic data standards to promote data interchange and reduce confusion and errors. To eliminate unnecessary misinterpretation of data, the PCWG strongly encourages that liberal documentation describing the standards and conventions used be attached to all PDS data products.

Both the adoption of the standards and the documentation of these standards and of the conventions associated with the data sets are important.

#### 7.6.7 References

The following two references give more detail on the cartographic data standards:

- (1.) Davies, M.E., *et al* (1986) Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1985 *Celestial Mechanics* **39**, 103-113.
- (2.) Batson, R. M., (1987) Digital Cartography of the Planets: New Methods, its Status and Future. *Photogrammetric Engineering & Remote Sensing* **53**, 1211-1218.

## Chapter 8

### TOOLS

A considerable quantity of software is being or has been developed under PDS support for use external to the system. There are routines that have been built to handle specific data sets. There are programs to expedite data interchange and to deal with database management.

The software exists in various states of development, from simply planned to completed. The list that follows is intended to guide the user in locating items of interest. Further information will be provided in subsequent versions of this document and can also be found by interrogating the PDS online catalog, once it is operational; the authors should be contacted regarding current status.

#### 8.1 DATA SET SOFTWARE

The Navigation Ancillary Information Facility (NAIF), which is developing and promulgating the SPICE concept (see Section 7.5) provides a software toolkit that is the very essence of the SPICE approach - the software allows the user to take control of geometry information relevant to his/her experiment data. The toolkit allows manipulation of SPICE kernels and calculation of geometric information of interest. Other available software includes a documentation reader. Further information is available through the authors or directly from Charles Acton at JPL.

The Radiometry Testbed Node of PDS has developed software for the treatment of radiometry and spectroscopy data. These packages, known as the XG and SPECIO systems, provide access to data from certain experiments on Mariner 6, 7, 9, and Viking, with constrained searches based on geometry or the measured parameters themselves. The systems run under TAE and are designed for portability to VAX computers. For further information contact Robert Gurule or Hugh Kieffer at:

US Geological Survey  
2255 N. Gemini Dr.  
Flagstaff AZ 86001  
(602) 527-7038

The Reflectance Spectroscopy Subnode at the Planetary Geosciences Division of the Hawaii Institute of Geophysics is developing a data management system for spectral data that will run on an IBM AT. It is intended to be a replicable system, with hardware, software, and data sets readily duplicated at user sites. Contact Tom McCord at (808) 948-6488.

The Central Node of PDS has developed a program to manipulate and display digital image data from CDROM or magnetic disks on IBM PC, XT, AT or compatible machines. It was developed specifically to enhance the usage of PDS CDROMs by the community. The source code (in "C") can be ordered or delivered electronically. For further information contact Mike Martin at the address in the second chapter of this document.

The JPL Imaging Testbed Node has been developing image processing programs that run on IBM PC AT computers. These perform a wide variety of functions and were modeled on routines available initially only through the Image Processing Lab at JPL. For further information contact Mike Martin or Sue LaVoie.

## 8.2 PDS LABEL SOFTWARE

PDS labels can be created easily with any text editor and mated with data files. The recipients of labeled data sets may choose, however, to treat the labels with software that places label information into catalogs or displays or prints it. PDS personnel have been developing routines to extract label information and parsers that can act on the content.

Automated loading of catalog information from labeled data sets is an important PDS development activity. Routines to perform this function are being implemented at the Central Node at JPL. Of particular interest is the template approach, in which model labels for specific data sets are prepared for developers, who then "fill in the blanks" with the relevant catalog data. Please contact the PDS Central Node's Data Management Team for information.

## Appendix A

### DATA INGESTION FROM PRESENT AND FUTURE MISSIONS

The negotiation with a Flight Project leading to the approval of the Project Data Management Plan (PDMP) by PDS is lengthy and involves Flight Project, PDS, PSDSG, NSSDC, NASA Headquarters and other science participation. This procedure identifies the various steps involved, the groups and organizations involved and their responsibilities leading to the PDS approval of a Flight Project PDMP. This procedure insures proper science and data administration involvement in identifying the Flight Project data sets to be prepared for delivery to the PDS and allows the PDS to identify the resources needed to properly support these data sets.

#### A.1 DRAFT PDS MISSION DATA INTERFACE LIST

- (1.) The PDS Mission Interface Team (MIFT) obtains an in-depth listing and description of all known mission products from the Project Interdisciplinary Scientist for Data Management and Archive or equivalent Project position (referred to as IDS/DMA). MIFT, in conjunction with the PDS Data Administrator, produces a PDS resource estimate for each product.
- (2.) The PDS MIFT gives this data products list and supporting data to the PDS Project Scientist.
- (3.) The PDS Project Scientist distributes the MIFT supplied information to the PSDSG, the NSSDC and the appropriate PDS Node Scientists for review. The PDS Project Scientist, PSDSG, NSSDC and Node Scientists may contact each other, the PDS MIFT and members of the Project for additional information and clarification.
- (4.) The PDS Project Scientist convenes a meeting including the PSDSG, the PDS Project Manager, the NSSDC, participating Node Scientists, the PDS MIFT and invited members of the Project. This group produces the Draft Mission Data Interface List. This list reflects both scientific and resource management scrutiny and may include additional data products not identified by the Project if needed.

#### A.2 WRITING THE PROJECT DATA MANAGEMENT PLAN

- (1.) The *PDS Guidelines for Project Data Management Plans* (PDMP), JPL Document D-5111, is provided to the Flight Project prior to drafting the PDMP. The PDS MIFT submits the Draft List to the Project through the IDS/DMA for inclusion into the Draft PDMP. This Draft PDMP is distributed for review to the Project including the Project Science Group (PSG), the PDS, the NSSDC, the PSDSG, etc.
- (2.) The Project IDS/DMA, assisted by the PDS MIFT, presents this Draft PDMP to the Project PSG and to NSSDC. The PDS MIFT resolves all PDS Interface related issues raised within its level of authority. Some issues affecting resources and schedule would be resolved at a higher level.
- (3.) The PDS Project Manager convenes a meeting of all interested parties (the PSDSG, PDS Project Scientist, PDS Node Scientists, PDS MIFT, Project PSG members including the IDS/DMA, NSSDC members, etc.) to air all unresolved issues and make the final data set/product identification.
- (4.) This Final List is submitted to the Project by the PDS MIFT through the IDS/DMA for inclusion in the Final PDMP which is approved by the PDS Project Manager and the NSSDC Director.

### **A.3 MECHANISM FOR CHANGE OUTSIDE OF STEPS 1 AND 2**

The following mechanisms are in place for additional comment, criticism, or change of the PDS negotiation:

- (1.) Review the matter directly with the PDS Project Scientist and Manager and with the NSSDC Director for resolution.
- (2.) Review the matter with the Planetary Science Data Steering Group (PSDSG) for consideration. This group has direct review authority for the PDS and has direct recommendation access to the NASA Code EL PDS Program Manager.
- (3.) Review the matter with the Project IDS/DMA, Scientist or Manager for action by the NASA Program Manager and/or Scientist for the Project who have direct recommendation access to the NASA Code EL PDS Program Manager.

## Appendix B

### DATA RESTORATION PROCEDURES

The identification and prioritization of data sets to be restored are based upon the needs of existing and future flight projects as well as current and future data analysis programs. The PDS does not have the breadth of oversight or responsibility of the Solar System Exploration Projects and Programs to make this identification and prioritization on its own. Therefore, NASA Program management and the PSDSG, which is chartered to advise NASA on science data issues, play the lead role in this activity. This procedure describes the steps, groups involved and their responsibility leading to the selection of proposals for data restoration.

#### B.1 DATA RESTORATION PRIORITIZATION

- (1.) The NASA Code EL Planetary Science Data Steering Group (PSDSG) has the lead responsibility and is supported by the PDS Project Scientist, the PDS Node Scientists and the Flight Project Scientists. A Priority List is generated, based upon PSDSG provided philosophy and criteria, which identifies desired data to be restored by Body, Spacecraft and Instrument. Additional data discrimination of Classification and Observation Category may also be necessary. This list is reviewed and updated annually.
- (2.) The PSDSG gives the Priority List to the NASA Code EL PDS Program Manager.

#### B.2 NASA REQUEST FOR DATA RESTORATION PROPOSALS

- (1.) The NASA Code EL PDS Program Manager releases a request for data restoration proposals annually which includes the PSDSG provided Priority List as well as data restoration guidelines and standards supplied by the PDS Data Administration Plan.
- (2.) Proposals for data sets not on the Priority List will also be considered.
- (3.) NASA Code EL sends all submitted proposals to the PDS Project Manager.

#### B.3 SELECTION OF DATA RESTORATION PROPOSALS

- (1.) The PDS Project Manager heads a review board supported by the PDS Project Scientist, Node Scientists and the PSDSG to select proposals within PDS resources.
- (2.) The PDS Project Manager negotiates the contracts, which are let from JPL, with the data restoration proposers.

#### B.4 RESTORATION PROCESS

- (1.) The PDS Science Manager monitors the delivery of scheduled data products.
- (2.) Delivered data is validated by the PDS Central Node staff or designated Discipline Node personnel for completeness, continuity and standards, and data integrity. The details of this peer review process are given in Section 2.4.5.
- (3.) The PDS Project Manager provides the NASA Code EL PDS Program Manager with a performance assessment of all data restoration activities annually.





## Appendix C

### WRITING CONVENTIONS AND DOCUMENT STANDARDS

The writing conventions and document standards for documents accompanying data sets being submitted to PDS are still being developed. The conventions and standards for use by PDS personnel on system development documents can be found in the document *Planetary Data System Writing Conventions and Documentation Standards*, March 31, 1988.



## Appendix D

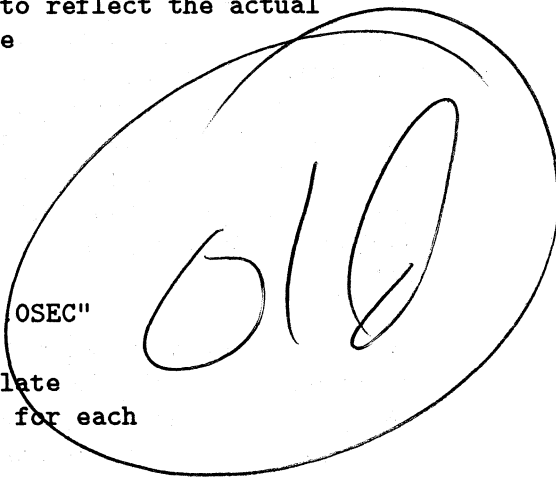
### DATA SET DOCUMENTATION EXAMPLES

The following material provides two complete examples of data set documentation for PDS data submission. The first was prepared for submission to PDS. The second was intended for use as a help file and is included as an example of content.

#### D.1 PLASMA WAVE DOCUMENTATION EXAMPLE

The following documentation was submitted during prototyping of the "template" process for automatic loading of catalog information for a data set. The Voyager PWS data set supplier provided information to fill in requested parts of the template.

```
/* Template:   PDS Dataset Catalog Input Template
/* Note:      The following templates form part of a standard
/*            set for the submission of a single dataset
/*            to the PDS.
/*            The following hierarchy was redone to reflect the actual
/*            templates completed for this example
/* Hierarchy: DATASETHLCAT
/*            DATASETINFO
/*            DSPARMINFO
/*            SCDATASET
/*            DSPROCESSING
/*
OBJECT          = DATASETHLCAT
DATA_SET_ID    = "VG2-U-PWS-2-S4.0SEC"
/*
/* Template:   Dataset General Information Template
/* Note:      This template is to be completed for each
/*            dataset cataloged in the PDS.
/*
OBJECT          = DATASETINFO
DATA_SET_NAME  = "VOYAGER 2 URANUS PLASMA WAVE RECEIVER
                EDITED SPEC 4.0SEC"
TARGET_NAME    = URANUS
TARGET_TYPE    = PLANET
START_EVENT_TIME = 1986-01-23T00:00:00.000Z
STOP_EVENT_TIME  = 1986-01-31T00:00:00.000Z
DATA_OBJECT_TYPE = "TIME SERIES"
RELEASE_DATE    = 1988-03-10
PROCESSING_TIME = 1988-02-14
PROCESSING_LEVEL_ID = 2 <CODMAC>
/* The following attribute is the dataset provider
FULL_NAME      = "DR WILLIAM S. KURTH"
INSTITUTION_NAME = "UNIVERSITY OF IOWA"
REFERENCE_KEY_ID = "N/A"
PIN_SOFTWARE_FLAG = Y
```



```

DETAIL_CATALOG_FLAG          = Y
/*
DATA_SET_DESCRIPTION         = "This data set consists of 4-second
edited, wave electric field intensities from the Voyager 2 Plasma Wave
Receiver spectrum analyzer obtained in the vicinity of the Uranian
magnetosphere. For each 4-second interval, a field strength is
determined for each of the 16 spectrum analyzer channels whose center
frequencies range from 10 Hertz to 56.2 kilo-Hertz and which are
logarithmically spaced in frequency, four channels per decade. Data
are edited, but not calibrated. Calibration look-up software and
tables are provided for use with this data
set."
/*
CONFIDENCE_LEVEL_NOTE       = "This data set includes all available
spectrum analyzer data available within the interval of time covered.
The data set has been cleaned as best possible for periodic noise
spikes due to a stepper motor operating on another experiment. Other
possible sources of noise which have not been eliminated include
random, bursty noise in the 178-Hertz channel due to impulsive noise
sources such as attitude control thrusters. The attitude control
thrusters also result in random noise spikes in all channels below 1
kiloHertz, with the most intense bursts occurring in the lowest for
channels, below about 60 Hertz. Also, a failure in the Voyager 2
flight data system a few months after launch has decreased the
sensitivity and the calibration accuracy of the upper 8 spectrum
analyzer channels (i.e. 1 kiloHertz and higher)."
END_OBJECT                  = DATASETINFO
/*
/* Template:Dataset Parameter Information Template
/* Note: This template shall be repeated for each
/* dataset, sampling parameter pair utilized
/* by a dataset in the PDS.
/*
OBJECT                      = DSPARMINFO
SAMPLING_PARAMETER_NAME     = TIME
SAMPLING_PARAMETER_RESOLUTION = 4.0
MINIMUM_SAMPLING_PARAMETER  = "N/A"
MAXIMUM_SAMPLING_PARAMETER  = "N/A"
SAMPLING_PARAMETER_INTERVAL = 4.0
MINIMUM_AVAILABLE_SAMPLING_INT= 4.0
SAMPLING_PARAMETER_UNIT     = SECOND
DATASET_PARAMETER_NAME      = "PLASMA WAVE SPECTRUM"
NOISE_LEVEL                 = 5.E-6
DATASET_PARAMETER_UNIT      = "VOLT/METER"
END_OBJECT                  = DSPARMINFO
/* Another Dataset Parmeter
OBJECT                      = DSPARMINFO
SAMPLING_PARAMETER_NAME     = "FREQUENCY"
SAMPLING_PARAMETER_RESOLUTION = .25

```

```

MINIMUM_SAMPLING_PARAMETER      = 1.0
MAXIMUM_SAMPLING_PARAMETER      = 4.75
SAMPLING_PARAMETER_INTERVAL     = .25
MINIMUM_AVAILABLE_SAMPLING_INT= .25
SAMPLING_PARAMETER_UNIT        = "LOG HERTZ"
DATASET_PARAMETER_NAME         = "PLASMA WAVE SPECTRUM"
NOISE_LEVEL                     = 5.E-6
DATASET_PARAMETER_UNIT         = "VOLT/METER"
END_OBJECT                      = DSPARMINFO
/*
/* Template: Spacecraft Dataset Template
/* Note:      This template shall be completed if the
/*            dataset is associated with a spacecraft.
/*
OBJECT                          = SCDATASET
SPACECRAFT_ID                   = VG2
INSTRUMENT_ID                   = PWS
END_OBJECT                      = SCDATASET
/*
/* Template: Dataset Processing Information Template
/* Note:      This template shall be completed for the
/*            most meaningful previous source dataset
/*            used to produce this dataset.
/*
OBJECT                          = DSPROCESSING
SOURCE_DATA_SET_ID              = "VG2-U-PWS-2-EDR"
SOFTWARE_NAME                   = "N/A"
END_OBJECT                      = DSPROCESSING
END_OBJECT                      = DATASETHLCAT
/* Template:PDS Catalog Spacecraft Instrument Input Template
/* Note:      The following templates form part of a standard
/*            set for the submission of a spacecraft instrument
/*            to the PDS.
/*            The following hierarchy was redone to reflect the actual
/*            templates completed for this example
/* Hierarchy: SCINSTINFO
/*            SCINSTOFFSET
/*            INSTELEC
/*            INSTDETECT
/*            INSTMODE
/*            INMODEPARM
/*            INMODEPARM
/*
OBJECT                          = SCINSTINFO
SPACECRAFT_ID                   = VG2
INSTRUMENT_ID                   = PWS
/* The following attribute applies to a NAIF dataset
DATA_SET_ID                     = "N/A"
INSTRUMENT_NAME                 = "PLASMA WAVE RECEIVER"

```

```

INSTRUMENT_TYPE           = "PLASMA WAVE SPECTROMETER"
/* The following attribute applies to the data provider
PDS_USER_ID               = WKURTH
/*
INSTRUMENT_DESCRIPTION    = "The Plasma Wave Receiver on Voyager
consists of both a 16-channel spectrum analyzer covering the range of
10 Hertz to 56.2 kiloHertz and a wideband waveform receiver which
returns the waveform of waves in the frequency range of 40 Hertz to 12
kiloHertz. The spectrum analyzer provides data on a continual basis
with a maximum temporal resolution of one spectrum per 4 seconds. The
waveform receiver returns 4-bit samples of the electric field measured
at a rate of 28,800 samples per second. Because of the very high data
rate, the waveform samples must be transmitted in the same manner as
the Voyager imaging information. At Jupiter, some 10,000 48-second
waveform frames were obtained. At Saturn and Uranus, the number of
frames obtained was very small due to the lower telecon rates available
at the greater distances of those planets."
/*
INSTRUMENT_CALIBRATION_DESC = "The Voyager plasma wave receiver
spectrum analyzers were calibrated by first establishing a relationship
between input voltage (of a sine wave at the filter center frequency)
and output voltage and second by measuring the effective bandwidth of
the filter. The bandwidth is measured by inputting a random noise
signal of known spectral density and by measuring the output voltage
which, by the first part of the calibration, is related to the rms
voltage of a sine wave. Dividing the equivalent sine wave voltage
squared by the input spectral density gives a bandwidth. This
procedure is repeated for each of the frequency channels. A special
calibration problem exists for the upper 8 frequency channels (1
kiloHertz and above) due to a failure in the Voyager 2 Flight Data
System. An in-flight recalibration was attempted using a Solar type
III radio burst observed by both Voyager 1 and 2. The recalibration
has known deficiencies, but it has been impossible to date to improve
on them."
/*
OPERATIONAL_CONSID_DESC    = "The primary operational considerations
of the PWS include maintaining the proper operating mode and obtaining
waveform samples as often as the spacecraft tape recorder/downlink
capabilities allow. The standard instrument mode is with Waveform
Power On and Input Gain State Hi. For encounter periods, this
corresponds to GS3GAINHI/WFMPWRON. Since there has never been a period
when the signal levels were so high as to require the Low input gain
state, and it is highly unlikely that such levels will ever be
encountered, Low Input Gain State should never be selected. As long as
there is power margin available, it is most straightforward to leave
the Waveform Receiver Power on. The power consumption is less than 0.5
Watt for this section, hence, the power savings afforded by turning it
off is not large. The most involved operational consideration is
providing for the transmission of waveform data to the ground. At

```

Jupiter, the majority of the waveform data could be sent directly to the ground via the 115200 bps downlink. This capability disappeared after Jupiter, however, because of the greater distance to the spacecraft, hence, lower telecom rates. Since operating the A/D converter at a rate less than 28800 Hertz would result in aliasing, it is necessary to record the data at the 115200 bps rate on the spacecraft tape recorder using the appropriate data mode and playback the recorded data at a lower rate, commensurate with the link capabilities. Again, choice of the proper playback mode is required. Since the data modes available on the spacecraft are highly dependent on mission phase, these modes are not described here."

/\*

SCIENTIFIC\_OBJECTIVES\_SUMMARY = "The primary science objective of the Voyager plasma wave investigation is to make the first surveys of the plasma and low frequency radio wave spectra in the magnetospheres of the outer planets: Jupiter, Saturn, Uranus, and Neptune. Plasma waves participate in a fundamental manner in the dynamics of planetary magnetospheres and in the interactions of that magnetosphere with the external solar wind and internal perturbations such as those induced by satellites interior to the magnetosphere. Plasma waves also provide diagnostic information about the plasma environment near the planets including such parameters as electron density and sometimes temperature. The instrument is also sensitive to low frequency radio emissions and, therefore, acts as a low frequency extension to the Planetary Radio Astronomy investigation. Radio waves are often the only means of remotely observing regions of plasma not accessible to the spacecraft and also lead to remote diagnostics of plasma conditions. The plasma wave receivers are also sensitive to the results of small dust particles impacting on various parts of the spacecraft at high velocities and, hence, provide a direct measure of the rate of impact, the density of the dust, and an estimate of the mass distribution of dust in the vicinity of the large planets, especially those with rings and otherwise dusty environments. Finally, the Plasma Wave Receiver will characterize the plasma wave and radio wave spectrum of the outer heliosphere and perhaps beyond, extending our understanding of solar wind plasma processes and wave-particle interactions to several tens of Astronomical Units."

/\*

INSTRUMENT\_HEIGHT = 4.8 <centimeter>  
INSTRUMENT\_LENGTH = 31.8 <centimeter>  
INSTRUMENT\_MANUFACTURER\_NAME = "UNIVERSITY OF IOWA"  
INSTRUMENT\_MASS = 1.4 <kilogram>  
INSTRUMENT\_SERIAL\_NUMBER = SNO03  
INSTRUMENT\_WIDTH = 18.5 <CENTIMETER>  
START\_TIME = UNKNOWN <build date>

/\*

/\* Template: Spacecraft Instrument Offset Information Template

/\* Note: This template is to be completed for each

/\* platform used for instrument positioning.

```

/*
OBJECT                = SCINSTOFFSET
PLATFORM_OR_MOUNTING_NAME = "SPACECRAFT BUS"
CONE_OFFSET_ANGLE     = "N/A"
CROSS_CONE_OFFSET_ANGLE = "N/A"
TWIST_OFFSET_ANGLE    = "N/A"
INSTRUMENT_MOUNTING_DESC = "N/A"
END_OBJECT            = SCINSTOFFSET
/*
/* Template:  Instrument Electronics Information Template
/* Note:     This template is to be completed for each
/*           instrument if applicable.
/*
OBJECT                = INSTELEC
/* Note:     Electronics id same as instrument id when no sub-system exists.
ELECTRONICS_ID        = "PWS"
/*
ELECTRONICS_DESCRIPTION = "The PWS electronics system consists of
three basic sections.  The first is the power supply system which
regulates and filters the 28 volt, 2400 Hertz spacecraft power supply
and provides DC voltages to the remainder of the instrument
electronics.  The second section is the spectrum analyzer which
consists of two banks of 8 narrow-band filters, each and two
logarithmic detectors, each of which provides an analog voltage
proportional to the log of the signal strength delivered to the
detector from any of the eight filters it services.  The analog outputs
from these two compressors, as they are called, are sent to the Flight
Data System of the spacecraft for conversion to an 8-bit digital value.
The spacecraft steps the inputs to the two compressors periodically
(once per 0.5 seconds in GS3 or encounter mode) so that signal
strengths in each of the 16 channels is measured over a 4-second
interval.  The third section consists of a single broadband filter of
40 Hertz to 12 kiloHertz, an automatic gain controlled amplifier, and a
4-bit A/D converter.  This section digitizes the electric field
waveform at a 28800 Hertz rate.  The output amplitude is controlled by
the automatic gain control in order to keep the signals within the
useful range provided by the 4-bit digitization."
END_OBJECT            = INSTELEC
/*
/* Template:  Instrument Detectors Information Template
/* Note:     This template is to be completed for each
/*           detector utilized by a instrument.
/*
OBJECT                = INSTDETECT
DETECTOR_ID           = "PWS ANTENNA"
DETECTOR_TYPE         = "DIPOLE ANTENNA"
DETECTORS              = 1
MAXIMUM_WAVELENGTH    = "N/A"
MINIMUM_WAVELENGTH    = "N/A"

```



```

NOMINAL_OPERATING_TEMPERATURE = 25 <c>
/*
INSTRUMENT_DETECTOR_DESC      = "The PWS uses a pair of 10 meter
antenna elements as a balanced dipole antenna. The two elements are
extended from the spacecraft at right angles to each other. (The
elements are shared with the Planetary Radio Astronomy instrument,
which uses them as a pair of monopoles so that measurements of the
degree of right and left hand circular polarization can be made.) The
PWS measures the voltage difference between the two elements which,
when coupled with the effective length of the antenna system--7.07 m)
yields an electric field strength in units of volt/meter. The antenna
system has the usual dipole antenna pattern which yields nearly 4*pi
steradians in its field of view, although there is a range of fields of
view where the detector response drops dramatically as one expects from
a dipole pattern."
SENSITIVITY_DESCRIPTION      = "The PWS antenna, used as a balanced
dipole with an effective length of 7.07 meters gives a sensitivity to
fluctuating (wave) electric fields down to the range of 5.E-6
volt/meter."
TEMPERATURE_TRANSLATION_DESC = "N/A"
END_OBJECT                   = INSDetect
/*
/* Template: Instrument Mode Information Template
/* Note: This template is to be completed for each
/* mode that a instrument may be configured.
/*
OBJECT                        = INSTMODE
INSTRUMENT_MODE_ID           = "GS3GAINHI/WFMPWRON"
/* The follow 3 duplicate entries are being remodeled at this time.
DATA_PATH_TYPE               = "REAL-TIME PLAYBACK"
DATA_PATH_TYPE               = "RECORDED DATA PLAYBACK"
DATA_RATE                    = 32 <bps>
DATA_RATE                    = 115200 <bps>
SAMPLE_BITS                  = 8 <bits>
SAMPLE_BITS                  = 4 <bits>
FOV_SHAPE_NAME               = "DIPOLE"
FOVS                          = 1
GAIN_STATE_ID                = "HI"
HORIZONTAL_FOV                = "2*PI" ??? FLOAT
/* Next value may be horizontal_fov
HORIZONTAL_PIXEL_FOV         = "N/A"
INSTRUMENT_POWER_CONSUMPTION = 1.6 <watt>
SCAN_RATE                    = "N/A"
/* Next value may be verticle_fov
VERTICAL_FOV                 = "2*PI" ??? FLOAT
VERTICAL_PIXEL_FOV          = "N/A"
/*
INSTRUMENT_MODE_DESCRIPTION  = "The PWS instrument gain is high and
the waveform receiver power is on. This is the normal encounter

```

operating mode of the instrument and places it in its most sensitive input gain state with the waveform receiver section turned on. The fact that the waveform receiver power is on does not guarantee that waveform data is available. The spacecraft is in the GS-3 data mode which cycles the plasma wave spectrum analyzer so that a complete spectrum is obtained every 4 seconds."

/\*

/\* Template: Instrument Mode Parameters Information Template

/\* Note: This template is to be completed for each

/\* mode parameter a instrument utilizes.

/\*

OBJECT = INMODEPARM  
 INSTRUMENT\_PARAMETER\_NAME = "WAVE ELECTRIC FIELD INTENSITY"  
 INSTRUMENT\_PARAMETER\_UNIT = "VOLT/METER"  
 MINIMUM\_INSTRUMENT\_PARAMETER = 5.E-6  
 MAXIMUM\_INSTRUMENT\_PARAMETER = 5.E-1  
 NOISE\_LEVEL = 5.E-6  
 MINIMUM\_SAMPLING\_PARAMETER = "N/A"  
 MAXIMUM\_SAMPLING\_PARAMETER = "N/A"  
 MINIMUM\_AVAILABLE\_SAMPLING\_INT= 4.0 <sec>  
 SAMPLING\_PARAMETER\_INTERVAL = 4.0 <sec>  
 SAMPLING\_PARAMETER\_NAME = "TIME"  
 SAMPLING\_PARAMETER\_RESOLUTION = 4.0 <sec>  
 SAMPLING\_PARAMETER\_UNIT = "SECOND"

/\*

INSTRUMENT\_PARAMETER\_DESC = "A measured parameter equaling the electric field strength in a specific frequency passband (in MKS unit: volts/meter) measured in a single sensor or antenna."

END\_OBJECT = INMODEPARM

/\* Another sampling parameter for the above instrument parameter

/\* This is being re-modeled at this time by Karen Paularena and

/\* Steve Hughes

MINIMUM\_SAMPLING\_PARAMETER = 1.0  
 MAXIMUM\_SAMPLING\_PARAMETER = 4.75  
 MINIMUM\_AVAILABLE\_SAMPLING\_INT= .25 <sec>  
 SAMPLING\_PARAMETER\_INTERVAL = .25 <sec>  
 SAMPLING\_PARAMETER\_NAME = "FREQUENCY"  
 SAMPLING\_PARAMETER\_RESOLUTION = .25 <sec>  
 SAMPLING\_PARAMETER\_UNIT = "LOG HERTZ"  
 END\_OBJECT = INMODEPARM

/\* Another instrument mode parameter

OBJECT = INMODEPARM  
 INSTRUMENT\_PARAMETER\_NAME = "ELECTRIC FIELD COMPONENT"  
 INSTRUMENT\_PARAMETER\_UNIT = "VOLT/METER"  
 MINIMUM\_INSTRUMENT\_PARAMETER = 5.E-6  
 MAXIMUM\_INSTRUMENT\_PARAMETER = 5.E-1  
 NOISE\_LEVEL = 5.E-6  
 MINIMUM\_SAMPLING\_PARAMETER = 3.47E-5  
 MAXIMUM\_SAMPLING\_PARAMETER = 3.47E-5

```

MINIMUM_AVAILABLE_SAMPLING_INT= 3.47E-5 <sec>.
SAMPLING_PARAMETER_INTERVAL   = 3.47E-5 <sec>
SAMPLING_PARAMETER_NAME       = "TIME"
SAMPLING_PARAMETER_RESOLUTION = 3.47E-5<sec>
SAMPLING_PARAMETER_UNIT       = "SECOND"
INSTRUMENT_PARAMETER_DESC     = "A measured parameter equaling the
electric field strength (e.g. in milli-volts per meter) along a
particular axis direction."
END_OBJECT                     = INMODEPARM
END_OBJECT                     = INSTMODE
END_OBJECT                     = INSTINFO
/* Template:   PDS Catalog References Input Template
/* Note:       The following templates form part of a standard
/*             set for the submission of a Publication References
/*             to the PDS.
/*             The following hierarchy was redone to reflect the actual
/*             templates completed for this example
/* Hierarchy:  REFERINFO
/*             REFERAUTHOR
/*             REFERAUTHOR
/*
OBJECT          = REFERINFO
DOCUMENT_TOPIC_NAME = "A PLASMA WAVE INVESTIGATION FOR THE
                    VOYAGER MISSION"
JOURNAL_NAME    = "SPACE SCIENCE REVIEWS"
PUBLICATION_DATE = 1977
REFERENCE_DESCRIPTION = "SCARF, F. L., AND D. A. GURNETT"
REFERENCE_KEY_ID  = "SCARF 77"
/*
/* Template:   Reference Authors Information Template
/* Note:       This template is to be completed for each
/*             author associated with the publication.
/*
OBJECT          = REFERAUTHOR
FULL_NAME       = "F. L. SCARF"
END_OBJECT      = REFAUTHOR
/* another author
OBJECT          = REFERAUTHOR
FULL_NAME       = "D. A. GURNETT"
END_OBJECT      = REFERAUTHOR
END_OBJECT      = REFERINFO
END_OBJECT      = DATASETHLCAT

```

## D.2 IRS DATA SET EXAMPLE

The following text was generated as part of a data set restoration effort by the Radiometry Node of the Pilot PDS. This information was intended for use in an online VAX help file. Please note that only very limited information was available about some aspects of the experiment.

1 IRS

The Mariner 6 and 7 Infrared Spectrometer experiment.

## 2 INSTRUMENT

The IRS instrument is comprised of a 10-inch Dall-Kirkham telescope feeding a pair of circular-variable filter spectrometers. Channel 1 covers 4.0 to 14.3 microns, detected by a HgGe detector at 22K, cooled by a Joule-Thomson cryostat. Channel 2 covers 1.9-6.0 microns; detection is by a PbSe detector at 175K, with radiative cooling. Wavelength resolution is 0.5-1.0%.

For detailed information see the instrument paper in REFERENCES.

## 2 TEAM

Information accurate as of 1984:

Principal investigator for IRS:

Dr. George C. Pimentel

Dept. of Chemistry

Univ. of California, Berkeley CA 94720

(415) 642-6330

Co-investigators:

Dr. Kenneth Herr

Aerospace Corp.

El Segundo CA

(213) 648-5620

## 2 COVERAGE

Detailed information on coverage is available in the IRS data file as geometry for each spectrum footprint. The geometry is contained in a header preceding each spectrum. Summary information is given below.

	Mariner 6	Mariner 7
Arrival date of closest approach	July 31 1969	Aug. 5 1969
Julian date	2440433.7216	2440438.7089
Areocentric solar longitude (Ls)	199.8	202.8
Time of closest approach (UTC)	5h 19m 06.8s	5h 0m 49.5s
Altitude of closest approach	3428.91 km	3428.35 km
Tangential velocity of s/c at closest approach	8.03 km/sec	7.90 km/sec
Latitude of subsolar point at C/A	-8.12 deg	-9.30 deg
Longitude of subsolar point	303.32 deg	356.00 deg
Longitude of terminator at equator at C/A	33.32 deg	86.00 deg

The size of the instrument field of view (a slit) is 2.07 by 0.10 degrees, or 36.1 by 1.75 milliradians. Multiplying the latter set of numbers by the range for a given spectrum yields the footprint size, in the same units as the range.

NOTE: the Mariner 6 instrument returned 108 Channel 2 (near-IR) spectra; no Channel 1 data were returned because the Joule-Thomson cryostat failed to cool the detector. The Mariner 7 instrument operated successfully in both channels and returned 130 spectra in each channel.

## 2 DATA\_FORMAT

IRS data are arranged as a single set of 512 spectra; there are first 36 calibration spectra, followed by 122 Mariner 6 spectra, followed by 354 Mariner 7 spectra:

Record	Spectra Nos.	Channel	Instrument	Subject
1-22	(negative)	1,2	2 (Mar. 7)	BB calibration
23-36	(negative)	2	1 (Mar. 6)	BB calibration
37-158	14-290	2	1	space, Mars
159-323	2-274	2	2	space, Mars
324-512	2-274	1	2	space, Mars

Each spectrum is accompanied by a header containing spectrum number, instrument number, channel number, spacecraft clock count, geometry information, and target blackbody temperature in the case of calibration data. The header consists of 30 floating point numbers. The spectra are each a set of 740 floating point numbers, ranging in value between about -10. to +99.999. The header format is as follows:

Word	Item
1	Instrument number (1 = Mar. 6; 2 = Mar. 7)
2	Channel number (1 = 4-14 microns; 2 = 1.9-6 microns)
3	Spectrum number (neg. for calibration)
4	GMT hour
5	GMT minute
6	GMT second
7	Encounter relative time: minutes
8	Encounter relative time: seconds
9	Spacecraft clock count
10	Latitude of slit center intercept on planet
11	Longitude of slit center intercept on planet
12	Latitude of slit north end intercept on planet
13	Longitude of slit north end intercept on planet
14	Latitude of slit south end intercept on planet
15	Longitude of slit south end intercept on planet
16	Emission angle (degrees) for slit center intercept
17	Incidence angle (degrees) for slit center intercept
18	Phase angle (degrees) for slit center intercept
19	Angle (degrees) between slit center intercept and planet center as seen from spacecraft
20	Time past local sunset at slit center intercept (hours and tenths)
21	Slant range (km) to slit center intercept
22	Altitude of tangent ray above planetocentric sphere for non-intercepting views
23*	Wavelength interval per spectrum point, left segment (between left and central spike)
24*	Wavelength (microns) of first point in left segment
25*	Wavelength interval per spectrum point, right segment (between central and right spike)
26*	Wavelength (microns) of point 350 in spectrum
27-30	Spare
*	These items have been derived from the spectra and were not included originally in the data set. See DECALIBRATION.

## 2 CALIBRATION

Pre-flight calibration of the IRS instruments consisted of obtaining spectra of blackbody sources at varying temperatures in the range 77–300K, as well as absorption spectra of NH<sub>3</sub>, CH<sub>4</sub>, H<sub>2</sub>O, CO<sub>2</sub>, and polystyrene with a high-temperature source. See the instrument paper under REFERENCES. Several blackbody calibration spectra at various target temperatures are available in the data file; they are the first spectra in the set, flagged by having negative spectrum numbers.

## 2 DECALIBRATION

Wavelength calibration information in flight was derived from observations of polystyrene film, which has many absorption features in the range of interest. The polystyrene spectrum was superimposed on the target (Mars) spectrum each 12th time. Unfortunately, these spectra are unavailable at the present time in digital form. The spikes that occur in all the spectra are introduced by allowing broadband radiation onto the detector at certain rotational positions of the circular variable filters. Ostensibly, these spikes represent wavelength fiduciarities. Thus, for Mariner 6(7), the spikes mark 1.88(1.88) microns and 3.72(3.69) microns for the shortwave part of channel 2, and 3.04(2.99) and 6.14(6.00) microns on the longwave side. Channel 1 is similarly marked; the first spike is at 3.89(3.86) microns, the middle spike at 7.92(7.88). The longwave segment of channel 1 uses the middle spike as 7.21(7.37) microns and the third spike as 14.69(14.45). However, it will be found that the spikes do not relate reproducibly to the locations of atmospheric CO<sub>2</sub> features; there is a small jitter in the relative locations. Wavelength information can also be derived from the known positions of these CO<sub>2</sub> atmospheric features in the Mars data; this is probably the most accurate and dependable scheme. However, the only sharp CO<sub>2</sub> features occur in the shortwave segment of channel 2 spectra. We have used the 2.0 and 2.69 micron features to derive the wavelength scale for this segment when possible; the parameters thus found are stored in header locations 23 and 24. For all other spectra, the wavelength parameters are found by extracting the location of the spikes and assigning them the wavelengths noted above (from the instrument paper).

Flux decalibration was performed by in-flight observations of an internal blackbody every 12th spectrum. The sequence of data acquisition was: BB calibration, 5 Mars spectra, polystyrene + Mars spectrum, 5 Mars spectra, BB calibration, etc. There was no significant change in photometric performance in flight, as measured by blackbody calibrations.

Although information is present in the instrument paper (figure 5) that would permit decalibrating the intensity scale, and detector response curves are also available for detectors of the kind used, we have not applied such corrections to the spectra because of the resulting uncertainties. For many purposes the overall shape of the spectra is of lesser importance; the thermal region spectra have good blackbody calibration spectra available that will permit assessment of instrumental effects (these are the first spectra in the file).

## 2 GEOMETRY

Position of the IRS FOV relative to the television experiment:

	Cone	Cross-cone
TV-B	-0 deg. 1'50"	+0 deg. 0'30"
TV-A	+0 14'50"	+0 9'31"
IRS	-3 53'12"	+1 33'42"

It is not stated whether these values apply to both spacecraft.



Groundtrack geometry information for the IRS data set is contained in the data file together with the spectra themselves. See DATA\_FORMAT for specific information. Available parameters are:

GMT TIME;            TIME RELATIVE TO ENCOUNTER;  
LATITUDE:    NORTH, SOUTH ENDS OF SLIT AND CENTER  
LONGITUDE:    "    "    "    "    "    "  
EMISSION ANGLE AT SURFACE INTERCEPT (SLIT CENTER)  
SOLAR INCIDENCE ANGLE AT SURFACE INTERCEPT  
SOLAR PHASE ANGLE AT SURFACE INTERCEPT  
ANGLE FORMED BY INTERCEPT POINT, SPACECRAFT, AND PLANET CENTER  
TIME PAST LOCAL SUNSET AT INTERCEPT POINT  
RANGE OF SPACECRAFT FROM SURFACE INTERCEPT  
ALTITUDE OF TANGENT RAY ABOVE PLANETOCENTRIC SURFACE (WHEN  
INSTRUMENT IS NOT POINTED AT PLANET)

Original longitudes supplied were east longitude, in the NA3 system operative in 1969. We have converted the latitudes and longitudes to the post-Viking system, incorporating both the redefinition of zero longitude and the relocation of the Martian pole. Longitudes are now west longitudes. For details of the major changes that occurred after Mariner 9 (the Viking alteration is minor), see the article by de Vaucouleurs in REFERENCES.

## 2 SOFTWARE

The only software available is the small set of IDL routines developed in 1984-5 by T.Z. Martin as this data set was incorporated into the PDS. They permit printing the header information, printing individual spectra, and plotting spectra to a Tektronix graphics terminal.

IRSOP:                    a program to read the IRS data and print header contents or individual spectra.

IRSPLOT:                    a program to plot spectra to a graphics terminal. The spectrum, channel, and instrument number will be requested; it will be helpful to have available a printout of the headers obtained with IRSOP. Subsequent spectra can be overplotted on the first, in different colors if the capability exists. The program sends escape codes to inform the terminal emulation software (ESC140 on the NEC APC computer) to change color. These codes may be changed as desired to suit other configurations.

## 2 REFERENCES

The following references will be of value to the researcher interested in the IRS instrument and data set; these are the main papers published by the team.

1. "Mariner Mars 1969 IR Spectrometer"; K.C. Herr, P.B. Forney, and G.C. Pimentel, *Applied Optics* 11, 493 (1972).
2. "Evidence about hydrate and solid water in the Martian surface from the 1969 Mariner IR Spectrometer"; G.C. Pimentel, P.B. Forney, and K.C. Herr, *J. Geophys. Res.* 79, 1623 (1974).
- 3 "Martian topography from the Mariner 6 and 7 IR spectra"; K.C. Herr, D. Horn, J.M. McAfee, and G.C. Pimentel, *Astron. J.* 75, 883 (1970).
- 4 "Evidence for solid carbon dioxide in the upper atmosphere of Mars"; K.C. Herr and G.C. Pimentel, *Science* 167, 47 (1970).
5. "The composition of the Martian atmosphere: minor constituents"; D. Horn, J.M. McAfee, A. M. Winer, K.C. Herr, and G.C. Pimentel, *Icarus* 16, 543 (1972).
6. "IR absorptions near three microns recorded over the polar cap of Mars"; K.C. Herr and G.C. Pimentel, *Science* 166, 496 (1969).

The following final report for JPL contract 951722 contains the best known geometry information for the data set and certain information about calibration. It is identical to the microfiched information available from the NSSDC:

"Infrared Spectrometer Mariner Mars 1969 - Data Format Report"; May 1 1970, G.C. Pimentel and K.C. Herr, University of California Space Sciences Laboratory Series 11 Issue 44, Berkeley, California 94720.

Reference for the old and new latitude/longitude system for Mars:

"Mariner 9 areographic coordinate system"; G. de Vaucouleurs, M.E. Davies, and F.M. Sturms, *J. Geophys. Res.* 78, 4395 (1973).

## Appendix E

### ENTITY DEFINITIONS AND STRUCTURES

This Appendix defines and presents the structure of each data entity contained in the PDS Entity-Relationship model. For each entity, a general textual description is followed by a data structure chart in indented-list format. Each structure chart identifies by name all of the groups and elements which comprise a particular entity and shows the hierarchical relationship between the components of the entity. Where an entity includes multiple occurrences of a group or an element, the designator (m) appears with the name. The definitions of groups and elements can be found in the PDS Data Dictionary. This appendix supersedes the entity structure found in the preliminary *Planetary Data System Data Dictionary*, D-4854, which was published on January 15, 1988.

#### Entity Name

---

- coordinate system information
- data set and product information
- earth bases information
- event information
- institution information
- instrument information
- instrument host information
- mission information
- node information
- parameter information
- personnel information
- platform information
- reference information
- software information
- spacecraft information
- target body information

## E.1 COORDINATE SYSTEM INFORMATION

The coordinate-system-information entity defines a reference coordinate system.

Level	Group/Element Structure
1	coordinate system id
1	coordinate system name
1	coordinate system center name
1	coordinate system ref epoch
1	coordinate system description
1	vector component information group (m)
2	vector component id
2	vector component type
2	vector component type desc
2	reference target name
2	reference object name
2	unit

## E.2 DATA SET AND PRODUCT INFORMATION

The data-set-and-product-information entity describes the contents and history of a data set or a data product. It uniquely identifies the data set or product as well as the producer of the data. It describes the data set or product in terms of its sampling, resolution, and physical or measured parameters and identifies references relating to the data set or product. It identifies the storage location and physical format of the stored data set or data product. In addition, where applicable and available, it provides information about the processing software and antecedent data used to produce the data set or product.

Note that for PDS Catalog purposes the notion of "data sets" includes the SPICE kernels produced by the Navigation Ancillary Information Facility (NAIF) at JPL.

Level	Group/Element Structure
1	data set identification group
2	data set name
2	data set id
2	data set description
2	target identification group (m)
3	target name
2	event time range
3	start event time
3	stop event time
2	data object type
2	release date
1	data set measurement group
2	measurement source desc
2	measurement atmosphere desc
2	measurement standard desc
2	measurement wave calbrt desc
1	data producer group
2	full name
2	institution name
2	processing level id
2	processing level desc
1	data processing history group (m)
2	source data set id (m)
2	software name
2	release date
2	processing time
1	confidence level note
1	document reference key id
1	pin software flag
1	detail catalog flag
1	data set instrument group (m)
2	instrument host id (m)
2	instrument id
1	reference key id (m)

### E.3 EARTH BASE INFORMATION

The earth-base-information entity describes earth bases and associates them with institutions and nodes.

Level	Group/Element Structure
1	earth base id
1	earth base name
1	node id
1	institution name

### E.4 EVENT INFORMATION

The event-information entity defines the approximate location and time of an observed event. It defines the location of the event with an initial position vector and, where appropriate, a spatial extent.

Level	Group/Element Structure
1	event identification group
2	event name
2	event type
3	event type description
2	target name
2	start event time
2	data set id
2	stop event time
2	instrument id
2	instrument host id
1	event location group
2	spatial coverage group
3	maximum latitude
3	minimum latitude
3	maximum longitude
3	minimum longitude
2	position vector group
3	coordinate system id
3	vector component id 1
3	vector component 1
3	vector component id 2
3	vector component 2
3	vector component id 3
3	vector component 3
3	local hour angle

## **E.5 INSTITUTION INFORMATION**

The institution-information entity provides the name of an institution.

<b>Level</b>	<b>Group/Element Structure</b>
--------------	--------------------------------

---

1	institution name
---	------------------

## E.6 INSTRUMENT INFORMATION

The instrument-information entity describes the characteristics of an instrument, including its mounting location, the scientific objectives for which it was designed, the observations in which it took part, the physical parameters measured by the instrument and sampling information required for the interpretation of instrument data values.

Each instrument consists of a set of subsystems. These subsystems are the detectors, optics, electronics and other components which define the system used to gather data. For a spacecraft-based instrument the set of subsystems is fixed, although only some parts of it may be used for any given observation (for example, only one of two available detectors may be in use at a particular time). This fixed set of subsystems which forms a spacecraft-based instrument is identified by a serial number, a name, and an identification. Many laboratory- or observatory-based instruments, however, do not consist of a fixed set of subsystems but are defined by the set of subsystems in use when particular data observations were made. In these cases, there is no defined instrument-specific identifying information other than the type of the instrument and the name of the laboratory and institution.

Level	Group/Element Structure
1	instrument identification group
2	instrument id
2	instrument name
2	instrument host id
2	instrument host type
2	instrument type
1	instrument desc
1	pds user id
1	data set id
1	start time
1	instrument data set group (m)
2	instrument parameter name
2	data set parameter name
2	important instrument parms
1	scientific objectives summary
1	instrument calibration desc
1	instrument mode group (m)
2	instrument mode id
2	gain state id
2	data path type
2	instrument power consumption
2	instrument mode desc
2	section id (m)
1	instrument section information group (m)
2	section id
2	horizontal pixel fov
2	vertical pixel fov
2	horizontal fov
2	vertical fov
2	fov shape name



- 2 data path description
- 2 data rate
- 2 scan rate
- 2 sample bits
- 2 fofs
- 2 filter number (m)
- 2 detector id (m)
- 2 telescope id (m)
- 2 electronics id (m)
- 1 filter group
  - 2 filter name
  - 2 filter number
  - 2 filter type
  - 2 minimum wavelength
  - 2 maximum wavelength
  - 2 center filter wavelength
  - 2 measurement wave calbrt desc
- 1 instrument detector group (m)
  - 2 instrument detector desc
  - 2 detector id
  - 2 detector type
  - 2 detector aspect ratio
  - 2 minimum wavelength
  - 2 maximum wavelength
  - 2 nominal operating temperature
  - 2 sensitivity desc
  - 2 temperature translation desc
- 1 instrument optics group (m)
  - 2 optics desc
  - 2 telescope group
    - 3 telescope id
    - 3 telescope diameter
    - 3 telescope f number
    - 3 telescope focal length
    - 3 telescope resolution
    - 3 telescope serial number
    - 3 telescope t number
    - 3 telescope t number error
    - 3 telescope transmittance
- 1 spacecraft mounting information group (m)
  - 2 platform or mounting name
  - 2 instrument mounting desc
  - 2 cone offset angle
  - 2 cross cone offset angle
  - 2 twist offset angle
- 1 earthbase mounting information group (m)
  - 2 platform or mounting name
  - 2 latitude
  - 2 longitude

- 2 instrument mounting desc
- 1 instrument physical characteristics group
  - 2 instrument mass
  - 2 instrument length
  - 2 instrument width
  - 2 instrument height
  - 2 instrument serial number
  - 2 instrument manufacturer name
- 1 instrument electronics group
  - 2 electronics id
  - 2 electronics desc
- 1 operational considerations desc
- 1 reference key id (m)

## E.7 MISSION INFORMATION

The mission-information entity provides a high-level description of a solar system mission or project and its objectives and characterizes each of its phases.

Level	Group/Element Structure
-------	-------------------------

- |   |                               |
|---|-------------------------------|
| 1 | mission name                  |
| 1 | mission desc                  |
| 1 | mission institution group (m) |
| 2 | institution name              |
| 2 | start time                    |
| 2 | stop time                     |
| 1 | mission objectives summary    |
| 1 | mission phase group (m)       |
| 2 | mission phase name            |
| 2 | mission phase type            |
| 2 | mission phase desc            |
| 2 | spacecraft id                 |
| 2 | target name                   |
| 2 | target type                   |
| 2 | start event time              |
| 2 | stop event time               |
| 1 | reference key id (m)          |

## E.8 NODE INFORMATION

The node-information entity provides information about a PDS Node. It identifies the Node Manager, the address of the Node and the institution at which the Node resides.

Level	Group/Element Structure
1	node name
1	node id
1	institution name
1	discipline name
1	node manager group
2	pds user id
1	node contact group
2	pds user id

## E.9 PARAMETER INFORMATION

The parameter-information entity provides information about instrument and data set parameters.

Level	Group/Element Structure
1	data set parameter group
2	data set id
2	sampling parameter name
2	sampling parameter resolution
2	minimum sampling parameter
2	maximum sampling parameter
2	sampling parameter interval
2	minimum available sampling int
2	sampling parameter unit
2	data set parameter name
2	noise level
2	data set parameter unit
1	instrument section parameter group
2	instrument id
2	instrument host id
2	section id
2	sampling parameter name
2	sampling parameter resolution
2	minimum sampling parameter
2	maximum sampling parameter
2	minimum available sampling int
2	sampling parameter unit
2	instrument parameter name
2	noise level
2	instrument parameter unit
2	minimum instrument parameter
2	maximum instrument parameter

## E.10 PERSONNEL INFORMATION

The personnel-information entity provides information about personnel associated with the PDS, including electronic and mailing addresses, a time-tagged affiliation history identifying the mission or task roles performed by a given person, the individual's institutional affiliations and his or her area of expertise.

Level	Group/Element Structure
-------	-------------------------

---

1	pds user id
1	full name
1	telephone number
1	fts number
1	mailing address line
1	discipline name
1	expertise area group (m)
2	expertise area id
2	expertise area desc
1	affiliation group
2	mission group (m)
3	spacecraft id
3	instrument id
3	mission name
3	role group (m)
4	expertise area id
4	specialty name
4	role name
4	start time
4	stop time
4	scientist funding id
2	task group (m)
3	task name
3	role group (m)
4	expertise area id
4	specialty name
4	role name
4	start time
4	stop time
4	scientist funding id
2	institution group (m)
3	institution name
3	scientist funding id
3	start time
1	node id
1	start time
2	electronic mail group (m)
3	electronic mail id
3	electronic mail type

### E.11 REFERENCE INFORMATION

The reference-information entity identifies documents which are referenced elsewhere in the PDS. Along with the traditional bibliographic information needed to reference journal articles, it provides information which will assist in referencing non-journal documents such as JPL-internal or other institution-internal documents.

Level	Group/Element Structure
1	reference key id
1	document topic name
1	publication date
2	author group (m)
3	full name
1	journal name
1	reference desc

### E.12 SOFTWARE INFORMATION

The software-information entity provides information about software available through the PDS. This includes information about the input data requirements, the input and output parameters which may be chosen, the output data and their formats and the required hardware and operating system environment. It identifies each program and its algorithms, along with the person to contact for additional information or for copies of the software.

TBD

### E.13 SPACECRAFT INFORMATION

The spacecraft-information entity describes the characteristics of a given spacecraft, including spacecraft operational information, information about instrument platforms on the spacecraft, and reference information.

Level	Group/Element Structure
-------	-------------------------

---

1	spacecraft id
1	spacecraft name
1	spacecraft desc
1	launch date
1	platform information group (m)
2	platform or mounting name
2	platform or mounting desc
1	spacecraft operations group (m)
2	spacecraft operations type
2	start event time
2	stop event time
1	reference key id (m)

## E.14 TARGET BODY INFORMATION

The target-body-information entity provides information which characterizes a particular solar system target body, such as the Sun, a planet, a satellite, an asteroid, or a comet. This includes various physical parameters and orbital parameters for the body. Along with each data value, this entity identifies the source of the data value, which, for example, may be a document, an institution or an individual.

Note that the most accurate ephemeris information for a given body is available in the appropriate SPICE kernel produced by the Navigation and Ancillary Information Facility (NAIF) at JPL.

Level	Group/Element Structure
1	target identification group
2	target name
2	target type
1	target physical information group
2	primary body name
2	rings
2	ring system summary
2	mean radius
2	a axis radius
2	b axis radius
2	c axis radius
2	flattening
2	mass
2	mass density
2	mean surface pressure
2	minimum surface pressure
2	maximum surface pressure
2	mean surface temperature
2	minimum surface temperature
2	maximum surface temperature
2	surface gravity
2	bond albedo
2	magnetic moment
1	target orbit group
2	rotation period
2	obliquity
2	pole right ascension
2	pole declination
2	synodic revolution period
2	sidereal revolution period
2	mean orbital radius
2	orbital semimajor axis
2	orbital eccentricity
2	orbital inclination
2	ascending node longitude
2	periapsis argument angle



2 secondary body name (m)  
1 target parameter information group  
2 target parameter name  
2 target parameter uncertainty  
2 target parameter epoch  
2 data source id  
2 data source desc



## Appendix F

### PDS CLASS AND DESCRIPTOR WORD DICTIONARY

#### F.1 INTRODUCTION

In the PDS naming syntax, the words forming a name are composed of specifiers (first, last, start, stop), descriptor words (which describe what is being measured or presented in the value field) and class words (which identify the gross data type of the object). Names are constructed using these word components from left to right, from most specific (the leftmost word) to most generic (the rightmost word). The following lists identify the current set of class words and descriptor words for use in PDS object naming. Appendix G provides a list of standard abbreviations for these words and other specifiers used in the PDS data dictionary.

#### F.2 CLASS WORD DICTIONARY

count	A numeric value indicating a current total or tally of an entity. The class word count is implied by the use of plural descriptor words such as lines, bytes or bits.  Example: LINES = 800 (interpreted as LINE_COUNT = 800).
date	A representation of time in which the smallest unit of measure is a day. The value is expressed in one of the standard forms.  Example: NATAL-DATE = 1959-05-30.
description	A textual account.  Example: "instrument-description"
flag	A boolean condition indicator, limited to two states.  Example: PRESSURE-VALVE-FLAG=TRUE.
group	Names a collection or aggregation of elements.  Example: IMAGE_IDENTIFICATION_GROUP.
id	A shorthand alphanumeric notation representing the common term used for an entity.  Example: SPACECRAFT_ID = VG1
mask	An unsigned numeric value representing the bit positions within an element value.  Example: SAMPLE_BIT_MASK = 2#00011111#.
name	A literal value representing the common term used to name an element.  Example: SPACECRAFT_NAME=MAGELLAN.

note	A textual expression of opinion, an observation, or a criticism; a remark.  Example: DATASET_NOTE = "This is a good dataset".
number	A number associated with an object.  Example: FILTER_NUMBER = 5
ratio	The relation between two quantities with respect to the number of times the first contains the second.  Example: SIGNAL_TO_NOISE_RATIO = 45.67
text	A free form text string of undefined content.  Example: OPERATIONAL_USAGE_TEXT = "Description of the operational usage of this instrument ...".
time	A value which measures the point of occurrence of an event expressed as date and time in one of the standard forms.  Example: HAPPY_HOUR_TIME=1987-06-21T17:30:30.0
type	A literal which represents a major predefined category.  Example: TARGET_TYPE=PLANET.
value	A numeric value representing a generic term for the amount or quantity of an entity where a more specific term is not defined. This is the default class word for names not terminated with a class word.  Example: SURFACE_TEMPERATURE = 98.6 would be interpreted as SURFACE_TEMPERATURE_VALUE.

### F.3 DESCRIPTOR WORD DICTIONARY

For Descriptor Words of a scientific nature (as opposed to the computer systems-oriented words such as "bits"), the definitions are intended to convey the meaning of each word within the context of planetary science and thus to facilitate the standardization of nomenclature within the planetary science community.

Certain descriptor words may have more than one meaning, depending upon the context in which they are used. It is believed that it is appropriate to include these words and their (multiple) definitions in the list, and that the context will suggest which definition is applicable in a given case.

In some cases (such as "elevation"), the usage example given for the Descriptor Word may contain just the word itself. In general, however, the Descriptor Word is one of several components of a data object's name.

"Plural Descriptor Words" are a special component of the PDS Nomenclature Standards. A list of these words follows the body of the Descriptor Words list.

Formerly used (or proposed) descriptor words which have been superceded by other words are also enumerated at the end of the main Descriptor Words list.

albedo	Reflectivity of a planetary surface or particle.  Example: "bond_albedo"
altitude	The distance above a reference surface measured normal to that surface. Note: see "elevation" and "height". Altitudes are not normally measured along extended body radii, but along the direction normal to the geoid; these are the same only if the body is spherical.  Example: "spacecraft_altitude"
angle	A measure of the geometric figure formed by the intersection of two lines or planes. Note: element definitions for angles should include origin and relevant sign conventions where applicable.  Example: "maximum_emission_angle"
axis	A straight line with respect to which a body or figure is symmetrical.  Example: "orbital_semimajor_axis"
azimuth	One of two angular measures in a spherical coordinate system. Azimuth is measured in a plane which is normal to the principal axis, with increasing azimuth following the right hand rule convention relative to the positive direction of the principal axis. PDS adopts the convention that an azimuth angle is never signed negative. The point of zero azimuth must be defined in each case.  Example: "sub_solar_azimuth"
bandwidth	The range within a band of wavelengths, frequencies or energies.  Example: "radar_bandwidth"
base	A quantity to be added to a value.
channel	A band of frequencies or wavelengths.
circumference	The length of any great circle on a sphere.
coefficient	A numeric measure of some property or characteristic.

component	1) The part of a vector associated with one coordinate. 2) A constituent part.  Example: "event_velocity_x_component"
constant	A value that does not change significantly with time.
consumption	The usage of a consumable.  Example: "instrument_power_consumption"
contrast	The degree of difference between things having a comparable nature.  Example: "maximum_spectral_contrast"
declination	An angular measure in a spherical coordinate system, declination is the arc between the Earth's equatorial plane and a point on a great circle perpendicular to the equator. Positive declination is measured towards the Earth's north pole, which is the positive spin axis per the right hand rule; declinations south of the equator are negative. The orientation of the Earth's equator must be specified; either the B1950 or J2000 reference coordinate system. PDS adopts J2000 as the default. (See also "right_ascension".)  Example: "declination"
density	1) The mass of a given body per unit volume. 2) The amount of a quantity per unit of space.  Example: "mass_density"
deviation	Degree of deviance.
diameter	The length of a line passing through the center of a circle or a circular object.  Example: "telescope_diameter"
distance	A measure of the linear separation of two points, lines, surfaces, or objects. See also "altitude," which refers to a specific type of distance. The use of the word "distance" supercedes the use of the word "range" as a measure of linear separation (see definition of "range" below).  Example: "slant_distance"
duration	A measure of the time during which a condition exists.  Example: "instrument_exposure_duration"

eccentricity	A measure of the extent to which the shape of an orbit deviates from circular.  Example: "orbital_eccentricity"
elevation	1) The distance above a reference surface measured normal to that surface. Elevation is the altitude of a point on the physical surface of a body measured above the reference surface; height is the distance between the top and bottom of an object. 2) An angular measure in a spherical coordinate system, measured positively and negatively on a great circle normal to the azimuthal reference plane. The zero elevation point lies in the azimuthal reference plane, and positive elevation is measured towards the direction of the positive principal axis. (See also "azimuth".)  Example: "elevation"
epoch	A specific instance of time selected as a point of reference.  Example: "coordinate_system_reference_epoch"
error	The difference between an observed or calculated value and a true value.  Example: "telescope_t_number_error"
factor	A quantity by which another quantity is multiplied or divided.  Example: "sampling_factor"
fov	(Acronym for "field_of_view") The angular size of the field viewed by an instrument or detector. Note that a field may require multiple field_of_view measurements, depending upon its shape (e.g., height and width for a rectangular field).  Example: "horizontal_fov"
flattening	A measure of the geometric oblateness of a solar system body, defined as the ratio of the difference between the body's equatorial and polar diameters to the equatorial diameter, or " $(a-c)/a$ ."  Example: "flattening"
fraction	The non-integral part of a real number. See "base".
frequency	The number of cycles completed by a periodic function in unit time.

gravity	The gravitational force of a body, nominally at its surface. Example: "surface_gravity"
height	The distance between the top and bottom of an object. Example: "scaled_image_height"
inclination	The angle between two intersecting planes, one of which is deemed the reference plane and is normally a planet's equatorial plane as oriented at a specified reference epoch. Example: "ring_inclination"
index	An indicator of position within an arrangement of items.
interval	1) The intervening time between events. 2) The distance between points along a coordinate axis. See also "duration" for time intervals. Example: "sampling_interval"
latitude	Multiple definitions exist for latitude. PDS looks to NASA's Planetary Cartography Working Group to provide specific recommendations for definition of this term. (See also "longitude".) Example: "minimum_latitude"
length	A measured distance or dimension. See also "height" and "width". Example: "telescope_focal_length"
level	The magnitude of a continuously varying quantity. Example: "noise_level"
line	1) A row of data within a two-dimensional data set. 2) A narrow feature within a spectrum. Example: "mailing_address_line_1"
location	The position or site of an object. Example: "document_location"



longitude	Differing definitions for planetocentric- and planetographic- longitude exist, and these definitions in turn depend on the definition of East or North. PDS looks to NASA's Planetary Cartography Working Group to provide specific recommendations for definition of this term. (See also "latitude".)  Example: "maximum_longitude"
mass	A quantitative measure of a body's resistance to acceleration.  Example: "instrument_mass"
moment	The product of a quantity (such as a force) and the distance to a particular point or axis.  Example: "magnetic_moment"
obliquity	Angle between a body's equatorial plane and its orbital plane.  Example: "obliquity"
parameter	A variable.  Example: "maximum_physical_parameter"
password	An alphanumeric string which must be entered by a would-be user of a computer system in order to gain access to that system.  Example: "account_password"
percentage	A part of a whole, expressed in hundredths.  Example: "data_coverage_percentage"
period	The duration of a single repetition of a cyclic phenomenon or motion.  Example: "rotation_period"
pressure	Force per unit area.  Example: "mean_surface_atmospheric_pressure"

right_ascension	(right_ascension) The arc of the celestial equator between the vernal equinox and the point where the hour circle through the given body intersects the Earth's mean equator reckoned eastward, in degrees. The Earth mean equator and equinox shall be as defined by the IAU as the 'J2000' reference system unless noted as the 'B1950' reference system.  Example: "right_ascension"
radiance	A measure of the energy radiated by an object.  Example: "spectrum_integrated_radiance"
radius	The distance between the center of and a point on a circle, sphere, ellipse or ellipsoid.  Example: "mean_inner_radius"
range	Numeric values which identify the starting and stopping points of an interval. Note: the use of the word "distance" supercedes the use of the word "range" as a measure of linear separation (see definition of "distance" above).  Example: "AXIS_n_BIN_RANGE" "emission_angle_range"
rate	The amount of change of a quantity per unit time.  Example: "nominal_spin_rate"
resolution	A quantitative measure of the ability to distinguish separate values.  Example: "sampling_parameter_resolution"
scale	A proportion between two sets of dimensions.  Example: "map_scale"
summary	An abridged description.  Example: "scientific_objectives_summary"
temperature	The degree or intensity of heat or cold as measured on a thermometric scale.  Example: "mean_surface_temperature"
title	A descriptive heading or caption.  Example: "sequence_title"

transmittance	The ratio of transmitted to incident energy. Example: "telescope_transmittance"
unit	A determinate quantity adopted as a standard of measurement. Example: "unit"
wavelength	The distance that a wave travels in one cycle. Example: "minimum_wavelength"
width	The distance between two sides of an object. See also "height" and "width." Example: "scaled_image_width"

#### F.4 PLURAL DESCRIPTOR WORDS

axes	The number of axes within a cube data object.
bits	A count of the number of bits within an elementary data item. Examples: "ELEMENT_BITS," "sample_bits"
bytes	A count of the number of bytes within a record, or within a sub-component of a record. Example: "RECORD_BYTES"
columns	A count of the number of distinct data elements within a row in a table.
detectors	A count of the number of detectors contained, for example, in a given instrument. Example: "detectors"
fovs	A count of the number of different fields of view characteristic of an instrument or detector Example: "fovs"
images	A count of the number of images contained, for example, in a given mosaic. Example: "mosaic_images"
items	A count of the number of data elements along a specified axis of a data array.

lines	A count of the number of data occurrences in an image array.
parameters	A count of the number of parameters in a given application. Example: "required_parameters"
points	A count of the number of points (i.e., data samples) occurring, for example, within a given bin. Example: "bin_points"
records	A count of the number of physical or logical records within a file or a subcomponent of a file. Example: "FILE.RECORDS"
rings	A count of the number of rings associated with a given solar system body. Example: "rings"
rows	A count of the number of data occurrences in a table.
samples	A count of the number of data elements in a line of an image array or a set of data. Example: "sequence_samples"
units	A count of the number of units of a particular type Example: "media_units"

## F.5 IDENTIFIED NON-DESCRIPTOR WORDS

The following words are not to be used as descriptor words. For each word, the list below explains why the word was not included in the descriptor words list and provides an alternative which is a recognized PDS descriptor word.

code	Ambiguous. Use "id" instead.
date/time	Unnecessary. Use "time" alone in naming fields which may carry both date and time information, or which carry only time information (i.e., fields which provide information in units not greater than hours). Use "date" alone only in naming fields which are to carry only date information (i.e., fields which provide information only down to the level of days).
definition	Unnecessary. Use "description" instead.
divisor	Unnecessary. Use "factor" instead.

field_of_view	Awkward. Use "fov" instead.
identification	Too long. Use "id" instead.
increment	Unnecessary. Use "interval" instead.
indicator	Unnecessary. Use "id" or "state" instead.
information	Ambiguous. Use "description" instead. (Note: "information" is used as a descriptor word in the names of Data Dictionary entity names on an exception basis).
mode	Unnecessary. Use "description" or "id," as appropriate, together with mode (e.g., mode_description or mode_id).
multiplier	Unnecessary. Use "factor" instead.
comment	Unnecessary. Use "note" instead.
order	The descriptor word should be id, type, or description, as in storage_order_description.
origin	The descriptor word should be description or group, as in projection_origin_group.
position	Unnecessary. Use "location" instead.
right_ascension	Awkward. Use "ra" instead.



## Appendix G

### PDS ABBREVIATION LIST

The following is "Version A" (February 3, 1988) of the PDS Standard Abbreviations List, which replaces the November 12, 1987 Preliminary Version. Version A reflects the PDS Data Design Team's comments on the previous version. The Standard Abbreviations List is a component of the PDS Nomenclature Standards, which are explained in this document.

The Abbreviations List provides one or more standard abbreviation(s) for every word which appears in any of the data object names in the PDS Data Dictionary, as well as for other words needed in PDS applications. Each abbreviation listed is unique. Where "(DROP)" appears in place of an abbreviation, the word is to be dropped - rather than abbreviated - when necessary in a given application.

The list is not expected to require any additional editing, except to ADD new words and their abbreviations, or to ADD new abbreviations for words already on the list. Thus, the list should be a trustworthy source of standard abbreviations for use throughout PDS Version 1.0 implementation.

The Abbreviations List is maintained by the PDS Data Management Team at JPL.

FULL WORD	ABBR #1	ABBR #2	ABBR #3
acceptance	acptnc	acpc	acp
account	acct	act	
address	addr	adr	
affiliation	affil	afil	
albedo	alb		
algorithm	alg		
altitude	alt		
and	(DROP)		
angle	ang	an	
anomaly	anom	anm	
antecedent	antcdt	ant	
approach	aprch	apch	apr
area	ar		
argument	arg		
ascending	ascndg	asc	
aspect	aspct	asp	
associated	assctd	ascd	asd
atmosphere	atmos	atms	atm
atmospheric	atmsc	atc	
author	auth	aut	
authority	authty	athy	aty
availability	avlby	avl	av
available	avlbl	avbl	avb
average	avg		
axis	axs		
azimuth	az		

FULL WORD	ABBR #1	ABBR #2	ABBR #3
band	bnd	bd	
bin	bn		
bit	bt		
body	bod		
bond	bon		
brightness	brgtns	btns	
browse	brows	bws	bs
byte	byt		
calibration	calbrt	calb	cal
carrier	car		
catalog	cat		
center	ctr		
channel	chan	chl	ch
characteristic	char		
clarity	clrty	clar	clr
clock	clk		
closest	clst	cls	
code	cod		
comment	cmt		
component	comp	cmp	
compromise	cmprms	cmps	cps
computer	cmptr	cmpr	
condition	cond	cnd	
cone	cn		
confidence	confid	conf	cnf
consideration	consid	cnsd	csd
consumption	cns	csp	
contact	cntct	cont	ctt
contamination	contam	cntm	ctm
contrast	contr	cnr	
control	ctrl	ctl	
conversion	conv	cnv	
coordinate	coord	crd	cd
coordinator	coodr	crdr	crr
count	cnt	ct	
coverage	covrg	cvrg	cvg
cross	crs		
customer	cust	cus	
data	dta	da	
date	dt		
day	dy		
declination	dec		
defining	defng	defg	dfg
definition	defn	dfn	
delimiting	delim	dlim	
density	dnsty	dens	dns
description	desc	dsc	dc



FULL WORD	ABBR #1	ABBR #2	ABBR #3
detector	det		
diameter	diam	dia	
discipline	disc	dis	
distance	dist	dst	
document	doc		
duration	duratn	dur	du
dynamic	dynam	dyn	
earth	eth		
east	est		
eccentricity	ecc		
electronic	electnc	elec	elc
elevation	el		
emission	emiss	emsn	em
entry	entr	etr	
environment	env		
ephemeris	eph		
epoch	epc	ep	
error	err		
event	evt	ev	
experimenter	exprmtr	xptr	xpr
expertise	exprts	xpts	xps
exposure	exposr	exp	ex
facility	facil	facl	fac
factor	fact	fct	
feature	featr	fr	
field	fild	fil	
field-of-view	fov		
fields-particles	fp		
filter	fltr	flr	
first	fst		
flag	flg	fl	
flattening	flatng	fltn	fln
flood	fld	fd	
focal	fcl		
format	frmt	fmt	
frame	frm		
from	fr		
full	-		
function	func	fnc	
gain	gn		
geometry	gmtry	geom	gom
gravity	grav	grv	
group	grp		
guidance	guidnc	guid	gdn
hardware	hw		
height	ht		
hierarchy	hier	hir	

FULL WORD	ABBR #1	ABBR #2	ABBR #3
history	hist	hst	
home	hm		
horizontal	horiz	hor	
hour	hr		
hourly	hrly	hry	
identification	id		
image	img		
imaging	imng	imn	
implementation	impl	imp	
incidence	incdnc	inc	
inclination	incl	inl	
information	info	inf	
initial	init	ini	
inner	in		
input	inp		
institution	instn	ist	
instruction	instruc	istc	isc
instrument	instr	inst	ins
integrated	integd	itgd	itd
interval	int	iv	
item	itm		
journal	jrnl	jl	
julian	jul	ju	
kernel	knl		
key	ky		
laboratory	lab		
language	lang	lan	
last	lst		
latitude	lat		
launch	lnch	lch	
length	len		
level	lvl	lv	
light	lt		
limb	lmb		
line	lin		
list	lis		
local	lcl		
location	loc		
longitude	lon		
magnetic	magntc	mgnc	mgc
mail	ml		
mailing	mlng	mlg	
manager	mgr		
manufacturer	mfr		
map	mp		
mass	-		
maximum	max	ma	

FULL WORD	ABBR #1	ABBR #2	ABBR #3
mean	mn		
measured	meas	ms	
measurement	measmt	msmt	mst
media	med		
memory	memry	mem	
method	mthd	mth	
middle	mid		
midnight	mdnt	mdt	
midsequence	midseq	msq	
minimum	min	mi	
mission	msn		
mode	mod		
model	mdl		
moment	momnt	momt	mmt
mosaic	mos		
motion	motn	mot	
mounting	mtg		
naif	nf		
name	nam	nm	
native	natv	ntv	
navigation	navgtn	nav	
node	nod	nd	
noise	ns		
nominal	nom		
north	nor	no	
note	nt		
notebook	ntbk	nbk	
number	num		
object	obj		
objective	objctv	ojtv	oj
obliquity	obliq	oblq	obq
observation	obs		
observatory	obsvty	oby	
of	(DROP)		
offset	ofst	ost	
operating	oprng	optg	opg
operation	oprtn	op	
operational	optl	opl	
optics	optcs	opcs	opc
or	(DROP)		
orbit	orb		
orbital	orbtl	orbl	or
order	ord		
orientation	orientn	ortn	orn
outer	out	ot	
output	outp	otp	
page	pg		

FULL WORD	ABBR #1	ABBR #2	ABBR #3
parameter	parm	par	pm
password	pswd	pw	
path	pth		
peak	pk		
percentage	prctg	pctg	pct
periapsis	peri	pps	
period	per		
personnel	prsnnl	psnl	psl
phase	phs	ps	
physical	phys	phy	ph
pixel	pix		
planet	plnt	pla	
platform	pltfm	plat	plt
point	pnt	pt	
pointing	pntg	ptg	
pole	pol		
position	pos		
power	pwr		
precession	precesn	pcsn	pcn
preference	prefnc	pref	prf
pressure	pres	prs	
primary	prim	pri	
prime	pme		
principle	prncpl	pcpl	pcl
privilege	privlg	pvlg	pvg
process	proc	prc	
processing	procg	prcg	prg
producer	prdr	pdr	
product	prdct	prdt	prd
production	prdctn	prdtn	prn
profile	profl	prfl	prl
program	pgm		
programming	pgmg	pgg	
projection	prjctn	prjtn	prj
publication	pub		
quality	qual	qly	
query	qry		
radiance	radnc	rdnc	rdc
radiometry	rdmtry	rdm	
radius	rad		
range	rng		
rate	rat		
ratio	rto		
rationale	ratnle	rtl	
received	rcvd	rcd	
reference	ref	rf	
region	rgn		

FULL WORD	ABBR #1	ABBR #2	ABBR #3
registration	regis	reg	
release	relse	rels	rls
remote	remt	rmt	
request	rqst	rqs	
required	req		
requirement	rqmnt	rqmt	rq
research	rsrch	rsch	rsh
resolution	res	rl	
resonance	resnc	rsnc	rsc
reticle	retcl	ret	
revolution	rev	rv	
right-ascension	ra		
ring	rg		
role	rol		
rotation	rot		
sample	smp	smp	
sampling	samplg	samp	smg
satellite	sat		
scale	scl		
scaled	scl	scl	
scan	scn		
schedule	sched	sch	
scientific	sctf		
secondary	secdry	sdry	sdry
selection	sel		
semi	sm		
semimajor	smaj	smj	
sensitivity	sens	sns	
sequence	seqnce	seq	sq
serial	serl	srl	
series	ser		
set	-		
shape	shp		
sheet	sht		
shipping	shipg	shpg	spg
shutter	shtr	shr	
sidereal	sidrl	sid	
size	sz		
slant	slnt	slt	
software	sw		
solar	sol		
source	srce	src	
south	sou	so	
spacecraft	sc		
spacecraft-clock	sclk	sck	
spatial	spatl	sptl	stl
special	specl	spcl	spl

FULL WORD	ABBR #1	ABBR #2	ABBR #3
specialty	specly	spty	sty
spectral	spctrl	sprl	spr
spectroscopy	spctrsy	spsy	ssy
spectrum	spctrm	spct	spc
spin	spn		
stabilization	stabzn	stbn	sbn
staff	stf		
standard	std		
start	st		
state	stte	stt	
status	stat	sts	
stop	stp	sp	
storage	stor	str	
sub	sb		
subnode	sbnod	sbnd	
subsystem	ss		
subtask	sbtsk	stsk	stk
summary	sumry	smry	smy
supplier	suplr	supp	sup
support	suprt	spt	
surface	srfc	srf	
synodic	syndc	syn	
system	sys	sy	
target	trgt	tgt	
task	tsk		
telephone	phone	phn	
telescope	tlscp	tlsc	tl
temperature	temp	tmp	
time	tm		
title	ttl		
topic	topc	tpc	
total	tot		
translation	trnsl	tnsl	tnl
transmittance	tnsmtc	tmtc	tmc
triaxial	triaxl	trxl	txl
true	tru		
twist	twst	twt	
type	typ		
unit	unt	un	
usage	usg		
user	usr		
validity	vldty	vldy	vdy
value	val	vl	
vector	vect	vec	
velocity	veloc	vel	
vendor	vndr	vnd	
version	vrsn	vrs	

FULL WORD	ABBR #1	ABBR #2	ABBR #3
vertical	vert	vrt	
view	vw		
volume	vol		
wavelength	wave	wvl	wl
weight	wght	wgt	wt
west	wst	ws	
width	wid	wd	





## Appendix H

### STANDARD FORMATTED DATA UNITS

#### H.1 INTRODUCTION

The following is an introduction to the SFDU concept, by John Johnson and Ed Greenberg of JPL. More complete reference information is available in the "blue book": *Standard Formatted Data Units - Structure and Construction Rules*, Consultative Committee for Space Data Systems Publication #CCSDS 620.0-B-1. An example of the way PDS is currently implementing the SFDU is given in Appendix L.

#### H.2 INFORMATION TRANSFER

The Standard Formatted Data Unit (SFDU) concept provides the protocols needed to enable the transformation of the discipline-oriented data users of today into a distributed confederated world wide scientific information network of the future.

The SFDU Concept provides:

- (1.) a means for globally defining and identifying data products,
- (2.) a means for aggregating instances of science, ancillary and meta data into data products, and
- (3.) a means for administering the data product definitions and description to ensure their accessibility and understanding.

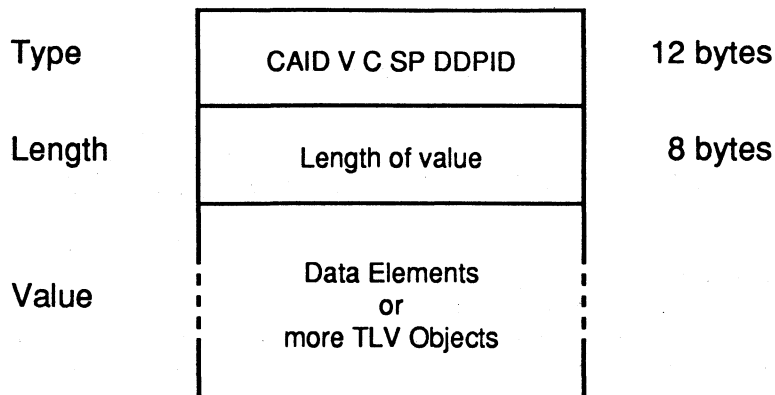
The SFDU methodology promotes documentation rigor through the administrative services provided by the CCSDS Member Agency Control Authorities (CA). These CAs thus become focal points for the acquisition of "meta data" (data about data). The data registration procedures establish a global data identification mechanism, which combined with standard data labelling and aggregation conventions, enables the comprehensive self identification process needed to support meaningful data interchange. The SFDU concept focuses on the standard *labeling* of data to enhance the transmission, storage, and manipulation of the data contained therein. The contents may be in *any* arrangement that can be expressed in a precise way.

The taxonomy of information transfer ranges from single data elements to completely identified and defined products. A data element is an individually named item of data that is used in a processing algorithm as a singular data parameter, variable, or attribute. Elements are collected and structured into data objects (aggregations of elements or groups) and units (aggregations of objects) with identifying SFDU labels. A data product consists of units containing not only the data, but data formats and representations, data element dictionaries, cataloging information, etc.

#### H.3 DATA STRUCTURING

A Type-Length-Value Object (TLVO) is a self identified and self delimited data object which follows CCSDS labelling rules. A structured TLV Data Object is shown in Figure H-1. It consists of a fixed length label followed by a variable length value field. The basic structure of the object is given in the figure below. The two fields of the label are: a) the TYPE field (which includes the reference name of the description of the value field) and, b) the LENGTH field (which provides the length of the value field). The value field may contain data elements or embedded TLVOs.

In the TYPE field, the Control Authority Identifier (CAID) identifies the CA office which maintains the format definition. The VERSION (V) field gives the structure of the label, the

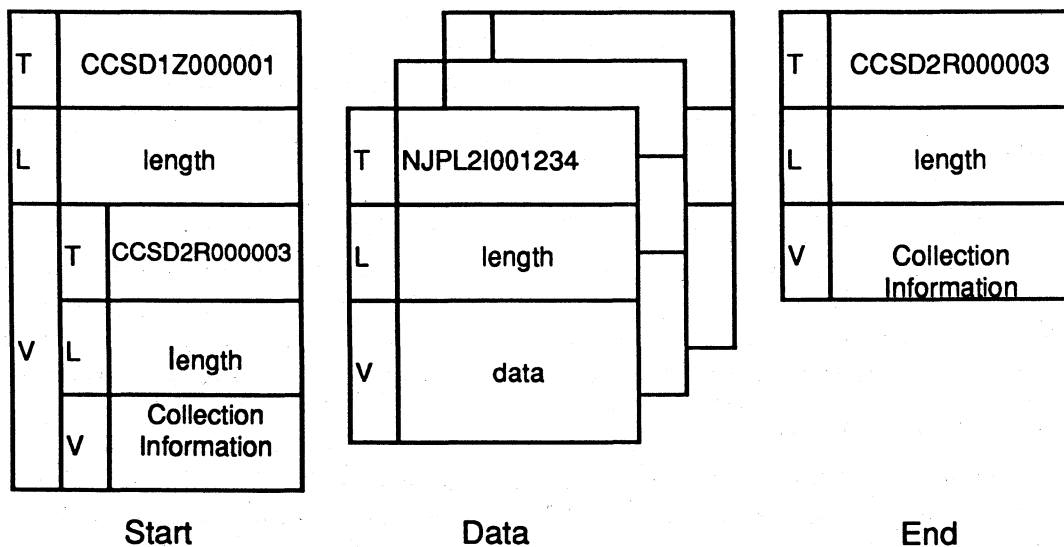


**Figure H-1: Structured TLV Data Object**

CLASS (C) field gives a high level classification of the content of the value field, and SP denotes two spare characters. The Data Definition Package Identifier (DDP ID) uniquely defines (within a CA) the package which contains the complete definition of the data object. The CA ID and The DDP ID together (called the Control Authority and Data Definition Package Identifier or ADI) provide globally unique identification and definition of the object.

Structuring (aggregation) of data is done currently in two ways: by envelope (length) and by reference. Envelope aggregation is depicted in the previous diagram; that is, the LENGTH field is known at the time of creation of the object. As shown, the value field may consist of data elements, in which case the ADI identifies the appropriate Data Description Package (DDP). Alternately, it may contain additional TLVOs. In other words, TLVOs may be nested arbitrarily deeply, forming data products or Standard Formatted Data Units (SFDU).

If a data product, consisting of several classes of objects where the total length is not known, is to be created, then aggregation by reference is used. An example of aggregation by reference is delimiting by marker. This is shown in Figure H-2.



**Figure H-2: Delimiting by Marker**

The diagram shows type (T), length (L), and value (V) fields of a set of objects making up a data product. The CA ID "CCSD" indicates that what follows is in SFDU format. Embedded

within the CCSD objects are Start and End (CLASS=R) labels, each of which describe the makeup of the object. The format of the R class value field is named as 0003, which within the CCSDS domain indicates a "keyword=value" format and specific semantics. The markers bracket a series of data objects (CLASS=I), the total length of which is not known at the beginning of product creation.

#### H.4 DATA DEFINITION

The Data Definition Package (DDP) structure and content is the subject of current study by the CCSDS. The intent is to supply the data product user with the conceptual or logical description of the data as well as the format and representation of the data. This information will be packaged with the data such that a suite of standard software, conforming to SFDU recommendations, at the user's installation can transform the data to conform to his machine architecture and can present standard views for applications.

The content of the DDP will include the following categories of information:

- (1.) Self identification information which contains the ADI of the TLVO whose value field is defined by this DDP. This may also include references to specific DEDs and DDRs.
- (2.) Data Entity Dictionaries (DED) which enable semantic information to be expressed.
- (3.) Data Definition Records (DDR) which contain the data object formats and representations which enable syntactic information to be interchanged among the elements of a heterogeneous information system.

The packaging of the DDP is shown in Figure H-3. In this example, a CLASS F object has been added to the beginning of the sample product shown previously.

This Class F Unit contains several embedded TLV objects each containing one of the sections of the DDP. In this way, the first logical piece of information received by the software suite is the identification and definition of the remainder of the product. One element of the DDP object is the CA ID/DDP ID of the data object that is defined. Thus the DDP can be loaded into a library and accessed whenever a CLASS I object of the same ID is received.

In the case of archives, the DDP information may be kept with the data and sent along with any order, enhancing the long term usefulness of the data.

#### H.5 TERMINOLOGY

The literature on SFDUs use a specialized vocabulary to describe the SFDU concept. The following definitions are provided to explain the terms.

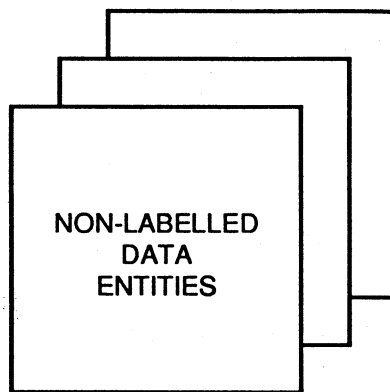
A DATA ENTITY is a logical collection of data that has a separate and distinct existence and objective. There are three types of data entities: data elements, data objects, and data products.

A DATA ELEMENT is the smallest named item or items of data for a given application.

TYPE-LENGTH-VALUE (TLV) is a method for the self-identification and delimiting of data. In this method, the TYPE identifies the specification governing the data within the VALUE. This specification establishes the order of appearance of the data elements with the data object and their data representation. The LENGTH expressed the size of the VALUE. The VALUE is the data. See Figure H-4.

The TYPE and LENGTH together form a LABEL while the VALUE field contains the data. The label and data together form a DATA OBJECT.

T	CCSD1Z000001	
L	LENGTH	
V	T	CCSD1F000001
	L	LENGTH
	V	EMBEDDED TLVOs FOR DED AND DDR INFORMATION
	T	CCSD1R000003
	L	LENGTH
	V	COLLECTION INFORMATION



DATA

T	CCSD1R000003
L	LENGTH
V	COLLECTION INFORMATION

END

START

Figure H-3: Product with DDP

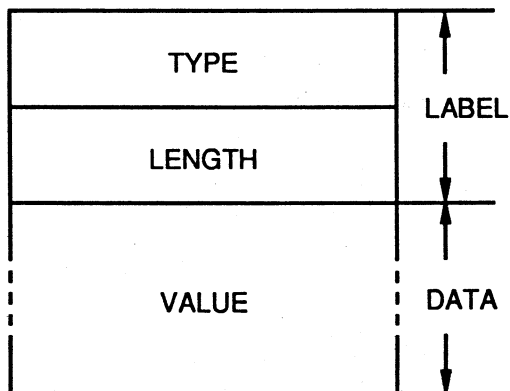


Figure H-4: TLV Encoded Structure

A TYPE-LENGTH-VALUE OBJECT (TLVO) is a TLV structured data object. A STANDARD FORMATTED DATA OBJECT (SFDO) is a TLVO that follows the specific CCSDS struc-

turing and labelling recommendations.

A CLASS UNIT is a collection of TLVOs that are aggregated for or by a specific application. A CLASS UNIT is recognized by the appearance of an ADI = CCSD0001 and any legal class ID.

A DATA PRODUCT or DATA UNIT is a collection of CLASS UNITS that are aggregated for transfer to or from a remote user process or archive. A STANDARD FORMATTED DATA UNIT (SFDU) is a data unit that consists of objects (and nested SFDU) aggregated by the CCSDS recommended construction rules. The start of a product is recognized by the appearance of an ADI = CCSD0001 and a class ID = Z.

Consider Figure H-5.

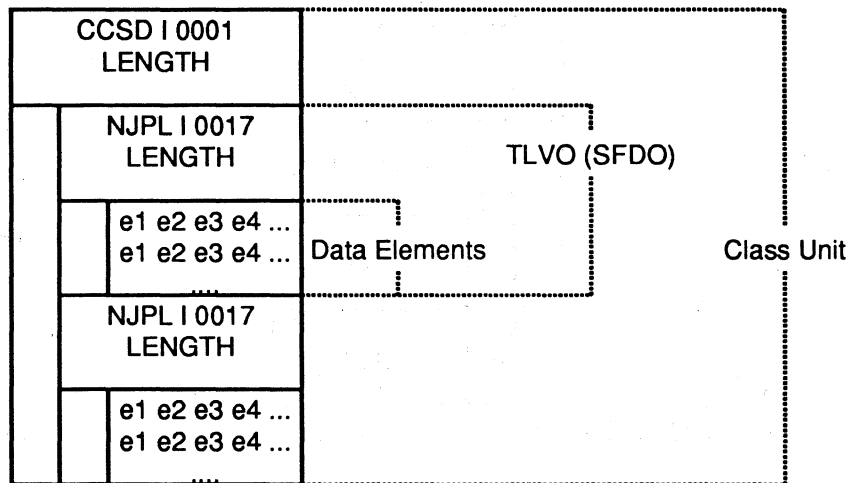


Figure H-5: Taxonomy

A DATA INSTANCE is a specific occurrence of values of a data entity.

The object depicted in Figure H-6, consists of two TLVOs containing instance identification information (typically used by a data catalog system) and data bits (perhaps an image). They are aggregated by the CCSD1I000001 label, forming a cohesive class unit. In each case, the length (L) field gives the length of the value (V) field. Note that a class unit typically contains all of the data objects necessary to process the science data, including calibration data, navigation data, etc. Data units range in complexity from simple messages, to entire collections of space acquired data plus ancillary and meta data from a mission.

The method described for formulating a data product for interchange is to assemble all of the required data in the desired order and construct an “envelope” or container that aggregates the combination, binding the enclosed data into a named and delimited data product. It is required that the labeling technique utilized in the “envelope” be globally recognizable and interpretable to ensure that the contents of the data product are readable.

The CCSDS requires that all data products be labelled using a CCSDS approved and registered data description.

## H.6 ODL IMPLEMENTATION OF SFDU'S

There are still uncertainties involving the final implementation of the SFDU architecture. In order to minimize the effect of small changes in the SFDU's which will naturally occur over the development and implementation stage, PDS will use a minimal implementation of the SFDU

T	CCSD11000001	
L	24621	
V	T	NJPL1K002468
	L	601
	V	(CATALOG DATA)
	T	NJPL1100TLM1
	L	23980
	V	BITS

**Figure H-6: CCSDS I Class Unit Instance**

registration scheme for archival data products. This SFDU (shown below) will indicate that the data file uses ODL labels to present data descriptive information about the data unit.

This label will constitute the first label record of an ODL labelled file.

NJPL1100PDS1nnnnnnnn = PDS\_SFDU\_LABEL where:

NJPL	is the JPL control authority.
1	version id, indicating that the sfd length is represented as an ASCII string.
I	class id, indicating that this is an information or data object class of SFDU.
00	are reserved characters filled with ASCII zeros.
PDS1	is the data definition record identifier, identifying the SFDU format as being a STANDARD PDS LABELLED file, with embedded format specification statements. This format supercedes (and is a superset of) the PDS0 SFDU label on current data files.
nnnnnnnn	is the length of the file in ASCII numerals; if this value is unknown use 00000000.

## Appendix I

### ODL IMPLEMENTATION AND SPECIFICATION

#### I.1 IMPLEMENTATION OF PDS OBJECT DESCRIPTION LANGUAGE

This section provides a description of the implementation of the Object Description Language and a detailed specification for the language.

##### I.1.1 PDS Object Description Language

In the object-oriented approach the principal data entity that is stored and transported is the data object. Examples of data objects are images, spectra and maps. For each object we need to have the following information:

- (1.) A description of the format of an object in terms a scientist can understand (scan lines, samples, etc). This description is done at the object class level and it is the same for every object in the class.
- (2.) A description of the content of the object. This description must be at the object instance level and will be different for every instance.

Data objects are encapsulated within data units for storage and transportation. For each data unit we need the following information:

- (1.) A description of the format of the data unit: is it a file or does it have some other organization? If it is a file, are the records fixed or variable length, how long are the records, etc.
- (2.) A description of the content of the data unit. If there are two or more data objects within the data unit and the descriptive information is the same for all the objects, then that descriptive information can be moved up to the level of description of the data unit.
- (3.) The location of each data object within the data unit.

All the descriptive information on data units and the objects within them is put into a "label" for the data unit. A label may be "embedded" - enclosed within the data unit itself - or "detached" - located in a separate label file that is associated with the data unit.

Figure I-1 shows the relationships between data units, objects, labels and the descriptive information about data units and objects contained within the label (The figure shows an "embedded" label).

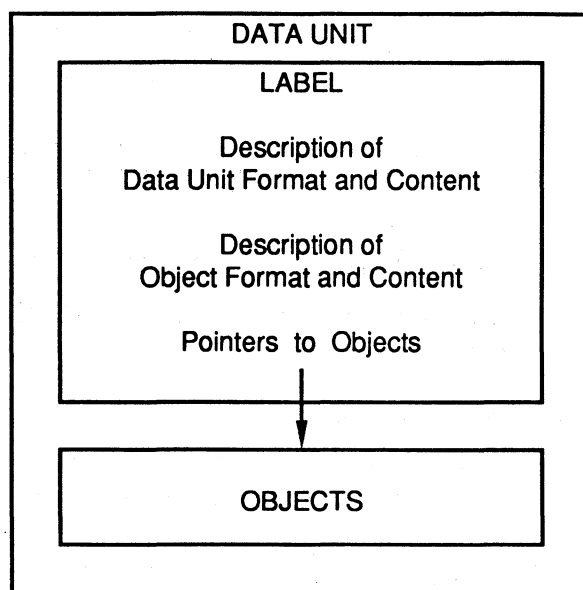
The PDS has developed the Object Description Language (ODL) to uniformly express information about objects and data units. Data unit labels, both embedded and detached, are created using this ODL. The ODL is used to describe objects and data units to both humans and computers so it is implemented much like a computer language: it is readable and writable by human beings but it can be readily parsed into a more efficient format which a computer can manipulate. We describe only the basics of the PDS Object Description Language in this section. A complete description of the syntax of the ODL is given in Section 2 of this Appendix.

An ODL object description has the following basic format:

OBJECT = object\_name

One or more statements in the Object Description Language

END-OBJECT



**Figure I-1: Structure of a Data Unit**

For descriptions of object instances, “object\_name” is a name assigned by the person creating the object description. It must be unique within the scope of the data unit within which the object is contained. Thus if there are three image objects within a data unit, each image must be assigned a different object name. This ensures that we can unambiguously identify each object within a data unit.

The statements within an object description all have the same general format:

name = value

where “name” is the name of a particular attribute of the object and “value” is the value of the attribute for this object instance. An attribute name can be 1 to 32 characters in length. The first character of a name must be alphabetic but the remaining characters, if any, may be any of the following:

Alphabetic characters. The ODL makes no distinction between uppercase and lower case alphabetic characters in names or anywhere other than within quoted text strings;

The decimal digits 0 - 9;

The underscore character (-).

The creator of a class of objects determines to a large degree which attributes are associated with those objects. Attribute names are not chosen arbitrarily; there is a PDS naming convention that specifies what an attribute name will look like and even which words to use in attribute names. These naming conventions are discussed in a Chapter 5.

Values in ODL statements can be either scalar (a single value) or array (a one or two dimensional set of values). Scalar data types are:

Integer	(example: 123)
Real	(example: 123.456)
Unitized real	(example: 123.456 <KM/SEC>
Literal	(examples: ORANGE, '1:1')
Text	(example: 'Now is the time')



Clock time           (example: 1987/10/26-22:03:45.6 <UTC>)  
Object names       (example: INTEGER, REAL, etc)

The ODL data types are described in greater detail in Appendix K.

Arrays are collections of scalar values separated by commas and optionally enclosed within parentheses. For example:

(123.0, 456.1, 789.2)

Here is an example of an object description for an instance of a histogram:

```
OBJECT = SAMPLE_HISTOGRAM
  ITEMS = 200
  ITEM_TYPE = INTEGER
  ITEM_BITS = 32
END_OBJECT
```

This description indicates that the particular histogram object known by the name SAMPLE\_HISTOGRAM contains the counts for 200 separate things and that the counts of these things are contained within an array of 200 integer values, with each integer value being 32 bits long. An equivalent FORTRAN 77 type declaration for this object would be:

```
INTEGER*4 SAMPLE_HISTOGRAM(200)
```

ODL descriptions of objects are used in two rather different ways: either they can describe an entire class of objects, thus serving as templates when individuals want to create objects of this class, or they can describe an instance of the object. We have so far discussed only the later case — where the description is for an instance of an object; a discussion of descriptions for object classes appears later in this section.

### **I.1.2 Data Unit Labels**

When objects are placed into data units the description of each object is placed into the data unit's label. For embedded labels, the label must be the first thing within the data unit. There are up to six parts to an embedded label, and by convention they appear in the order given here:

- (1.) SFDU registration
- (2.) Data unit format description
- (3.) Pointers to objects
- (4.) Data unit content description
- (5.) Object descriptions
- (6.) END statement

#### **I.1.2.1 SFDU Registration**

Most PDS data units will have a 20-byte Standard Format Data Unit (SFDU) registration ID in the first 20 bytes of the label (and hence the data unit). This SFDU ID is encoded as a statement in the ODL with the following format:

```
nnnnnnnnnnnnnnnnnnnn = SFDU_LABEL
```

where nn..nn is the 20-byte SFDU ID. SFDU IDs are issued by a special SFDU control authority and the rules for constructing SFDU IDs, as well as the procedure for getting an ID from a control authority, are discussed elsewhere (see Section 6.1 and Appendix E). The SFDU ID must begin

in the first column of the first line of the label (and the data unit). An example of an SFDU registration statement is:

```
NJPL1I00PDS100000000 = SFDU_LABEL
```

Additional descriptive material about the SFDU architecture is contained in Appendix H.

### **I.1.2.2 Data Unit Format Description**

A data unit's format description presents the salient characteristics of the data unit which must be known to properly open and read the data unit with a computer (usually through the computer's file management system). For example, the following ODL statements are used to describe the format of data units contained within files:

```
RECORD_TYPE    = FIXED or VARIABLE
RECORD_BYTES   = length of record (maximum length for variable records).
FILE_RECORDS   = number of records in file
LABEL_RECORDS  = number of records in label
```

As you can see in the example above, the format of a data unit is described in the same way as the format of a data object: through a set of statements in the ODL. In fact, specific types of data units - like files - are nothing more than objects of class DATA\_UNIT. This reveals the true nature of data units: they are objects created expressly to hold other data objects during transport and storage. The class descriptions for data units are contained in object description libraries, along with the descriptions of other types of objects.

### **I.1.2.3 Pointers to Objects**

It is usually useful to have a pointer within a data unit to each of the data objects within the data unit. This permits more rapid retrieval of the individual data objects. These pointers are expressed in the ODL using the following notation:

```
^object_name = location
```

In a file the location is given as an integer representing the starting record number of the object, measured from the beginning of the file. Pointers are also useful for describing the location of individual components of a data object and for pointing to a detached label. An example of the latter is

```
^object_STRUCTURE = "file_name"
```

where "object" is the name of the object being described and "file\_name" is the name of the detached label file containing the description.

### **I.1.2.4 Data Unit Content Description**

A data unit may contain ODL statements that describe the content of the data unit and its objects. Typically these ODL statements are derived by factoring ODL statements about the content of the data objects within the data unit: if there are ODL statements describing the content of data objects that are common to all the data objects within the data unit, then these ODL statements can be moved out of the object descriptions and into the data unit description. An example would be an image and a histogram of that image packaged into the same data unit: rather than putting the same information about image content into both the image object and histogram object descriptions, it is sufficient to state that information once in the data unit description.

### **I.1.2.5 Object Descriptions**

A label contains an object description for each of the objects within the data unit. This will include information about both the format and content of the data object.

There are three ways to specify object descriptions in labels. Firstly, for objects of a non-varying class – a class where the object format is fully defined by the class template so that all objects of the class are identical in format – it is sufficient to identify the object by reference, that is by simply naming the class of the object in the following manner:

```
OBJECT = MIRANDA1
      CLASS = VOYAGER_IMAGE
END_OBJECT
```

This is an acceptable description of the format of the object, although the receiver of such an object would have to consult an object definition library to fetch the template giving the details on the format of objects of class VOYAGER.IMAGE.

Secondly, for objects from a class with varying attributes – a class where the values for some attributes are not fully specified in the class template and therefore must be specified for each and every instance – the alternative is to name the object class and then supply the values for all varying attributes:

```
OBJECT = SAMPLE_HISTOGRAM
      CLASS      = HISTOGRAM
      ITEMS      = 200
      ITEM_TYPE  = INTEGER
      ITEM_BITS  = 32
END_OBJECT
```

Values for any varying attributes of an object must be specified when a description of such an object is created, but at the discretion of the creator of an object description some or all of the non-varying attributes may also be specified in the description. Supplying all the attributes, both varying and non-varying, is good practice for data that is to be widely distributed because it means the object can be processed without first having to retrieve the object's class template from an object description library.

Lastly, if the object does not belong to a pre-existing class of objects you omit the reference to an object class and then include values for all attributes in the object description:

```
OBJECT = SAMPLE_HISTOGRAM
      ITEMS      = 200
      ITEM_TYPE  = INTEGER
      ITEM_BITS  = 32
END_OBJECT
```

### **I.1.2.6 END Statement**

Each label must end with an ODL statement of the form:

```
END
```

This statement signifies the end of the label; computer-aided label processing will always terminate at this statement.

### I.1.3 Accessing Data Objects

The PDS will develop and distribute some of the basic software needed for handling data objects and data object descriptions written in the ODL. In particular the PDS will supply software for:

- (1.) Opening a data unit
- (2.) Extracting the label from a data unit
- (3.) Parsing ODL statements in the label
- (4.) Providing information about the objects within a data unit
- (5.) Providing access to the data objects within a data unit

The software developed by PDS will enable data objects and data units to be manipulated in three environments:

- (1.) *Human User Environment* – This is the simplest kind of object processing, where we present the text of a data unit label to the user on a CRT screen and the user reads and “processes” the label information to determine the objects within the data unit and how to manipulate them. To facilitate this type of processing we have purposely made the ODL easy for humans to read and to understand.
- (2.) *Programming Environment* – This is where information extracted from labels is used within programs written by users. User programs will have access to the description of the data unit, to the descriptions of the data objects within the data unit, and to the data objects themselves. This capability will be implemented through a series of calls in the host language and perhaps through a pre-processor as well. The initial host language for this capability will be Fortran 77 and the initial implementation will be for VAX/VMS computer systems.
- (3.) *Smart Software Environment* – This is the environment where information in a label tells “smart” software how to process the data unit and the data objects within it. Two candidates for smart object processing software have been identified:
  - (a.) *Data ingest*: Software for PDS central and discipline nodes that reads the labels attached to data flowing into the node to determine how to transfer the data into the node catalog or database.
  - (b.) *Data display*: Software that uses labels to determine how to display the data to which the labels are attached. For example, the software might know to display spectra (or other single-dimension data) as a line graph on a graphics terminal and images (or other two-dimensional data) on an image display device.

The object access software to be developed is depicted in Figure I-2.

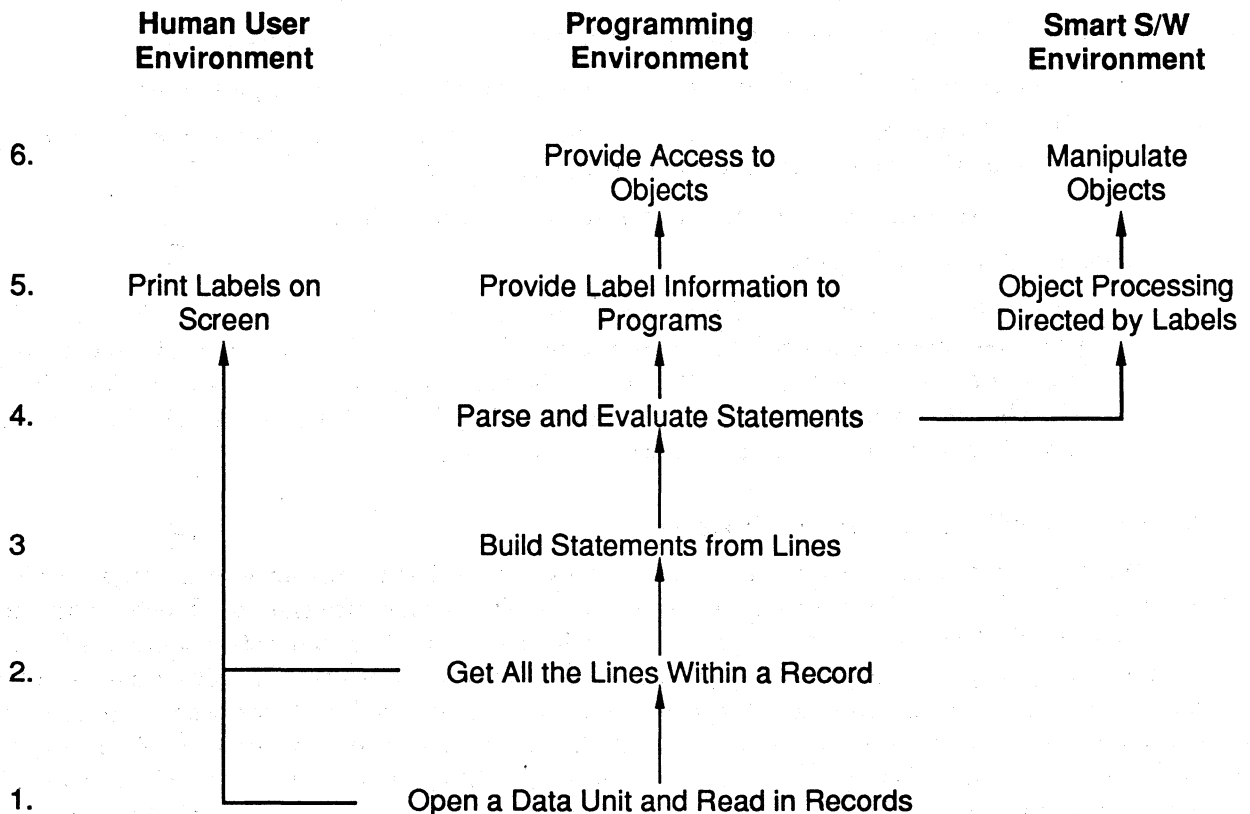
The processing is divided into six levels. Processing starts at level 1 and works its way to higher levels. You will note that the human user environment bypasses some of the higher levels because the brain is being called upon to do those jobs. Following is a description of each of the levels:

#### Level 1 – Open a Data Unit and Read in Label Records

This level opens a data unit, finds its label – either embedded or detached – for the data unit and reads in records containing label lines.

#### Level 2 – Get All the Lines Within a Record

In the case where labels are attached to the front of data it is often the case that multiple label lines are packed into a single record. For example, this is the case with the Voyager Image labels on the CD ROM where the records are 836 bytes long to match the size of



**Figure I-2: Object Access Software Layers**

the image records. In this case software is needed to strip out all the label lines in an input record. There must be an integral number of label lines in an input record.

#### Level 3 – Build Statements from Lines

The basic entity for label processing is the label statement. A label statement can take up a part of a line (there might be a comment on the end of the line) or it can span multiple lines. Software at this level builds statements out of one or more input lines.

#### Level 4 – Parse and Evaluate Statements

Once a statement has been assembled we can parse the statement. The result of parsing a label is:

- (i.) A list containing the name and value(s) for each attribute extracted from the data unit description portion of the label;
- (ii.) A list containing the name and class of each object within the data unit plus the pointer, if any, to the object;
- (iii.) For each object, a list containing the name and value(s) for each attribute extracted from the object description.

#### Level 5 – Provide Access to Data About Objects

At this level the information about objects gathered during parsing is made available to users. There are two ways in which this information will be provided:

- (i.) As output on a screen that provides users directly with information about the objects within the data unit;
  - (ii.) Through subroutine calls to user programs and smart software.
- Subroutines will be available to provide at least the following capabilities:
- (i.) Determining which objects are within a data unit (returns the number of data objects plus the name, class and pointer for each object);
  - (ii.) Getting the value for any attribute of a specified object.

#### Level 6 – Access to Objects

At this level access to the data objects within a data unit is provided. This capability is provided only through software subroutine calls. Subroutines will be available for retrieving all or part of an object so that it can be manipulated by the host program. Software will also be available for writing objects to a file, although this software will be available sometime after the retrieval software is completed.

#### I.1.3.1 Describing Classes of Objects

We noted above that object descriptions can define either an instance of an object (which is what they do when the descriptions appear within labels) or an entire class of objects. This section discusses object descriptions for object classes, which we will often call “class templates”. A new class template must be developed when a new class of objects is created. The class template is placed into an object description library where it can be retrieved by those who want to create a description for instances of that class. Essentially when you create an instance of an object you do so by taking the class template and, if necessary, filling in the blanks within the template. We expect that the creation of an object description from a class template will be done using a PDS-supplied computer program for writing labels.

Object descriptions for object instances and object classes share the same ODL format discussed previously. How then do you know whether a particular object description is to be interpreted as a template for a class of objects or as a description of an instance of an object? As with the distinction between the concepts of class and instance, the context helps make it clear. Any object description within a data unit label – either embedded or detached – is a description of an instance of an object. Object descriptions for classes of objects don’t appear in labels; they typically appear only within libraries of class descriptions that are accessible to people and software who are creating new objects or seeking further information on data objects they have received. Class templates contain other objects that define each attribute of the object. The general format of a class template is:

```

OBJECT = object_class
  OBJECT = first_attribute_name
    Description of first attribute
  END_OBJECT
  OBJECT = second_attribute_name
    Description of second attribute
  END_OBJECT
  .
  .
  .
END_OBJECT

```

The outermost object being defined here is the class template. The object class specified in the first OBJECT statement gives the name of the class of object and it must be unique within the PDS. Inside the class are other object descriptions that serve as attribute templates. There is one of these attribute templates for each attribute of the data object. Let's look at an example. We have previously shown the format description for an instance of class HISTOGRAM:

```
OBJECT = SAMPLE_HISTOGRAM
  CLASS      = HISTOGRAM
  ITEMS      = 200
  ITEM_TYPE  = INTEGER
  ITEM_BITS  = 32
END_OBJECT
```

Now let's look at the class template for class HISTOGRAM:

```
OBJECT = HISTOGRAM
  OBJECT = ITEMS
    TYPE      = INTEGER
    VALUE     = 1..*
    DESCRIPTION = "The number of bins within the histogram"
  END_OBJECT
  OBJECT = ITEM_TYPE
    TYPE      = OBJECT
    VALUE     = {INTEGER, REAL}
    DESCRIPTION = "The data type of the bins within the histogram"
  END_OBJECT
  OBJECT = ITEM_BITS
    TYPE      = INTEGER
    VALUE     = 1..*
    DESCRIPTION = "The length of each bin item, in bits."
  END_OBJECT
END_OBJECT
```

The name of the outermost object gives us the class name HISTOGRAM. Within the HISTOGRAM object are three attribute templates that provide information about the three attributes that are needed to describe a histogram. These three attributes are ITEMS, ITEM\_TYPE and ITEM\_BITS. In essence this class template says that you can specify any histogram by supplying values for these three attributes.

An attribute template always contains three ODL statements that define the attribute. These statements define the following:

- (1) *TYPE* – Specifies the type of value expected for this attribute. Must be one of the recognized ODL data types.
- (2) *VALUE* – The value or range of values for the attribute. If a single value is specified then it is the value for every instance of the object. If a range of values or a list of values is given then a value within the range of values (or selected from the list of values) must be specified for each object instance. An attribute with a single value is called “non-varying”; an attribute with a range or list of values is called “varying”. An object class with one or more varying attributes is called a varying object class. When a description for an instance of a varying object class is specified values for all varying attributes must be specified in the format description of the

object. All three of the attributes in our histogram example are varying, which says that they must appear in the definition for the object that we called SAMPLE\_HISTOGRAM.

- (3.) *STRUCTURE* – A text string containing an English description of the attribute. This text can be displayed by the software that builds object descriptions to the human user to help them understand what is expected of them when supplying an attribute value.

The PDS will build software that provides the following capabilities for creating and manipulating class templates:

- (1.) Creating an object class template: A program will query users for the name of each attribute and for the type, value and description of each attribute, building the attribute templates and the class template and, upon completion, inserting the class template into an object description library.
- (2.) Accessing an object description library and extracting a class template for a specified class of object.
- (3.) Parsing a class template for use by the software described below. It is expected that this will be the same parser that is used when processing data unit labels.
- (4.) Printing the description for a class of object using the class template as a guide.
- (5.) Creating a description of an instance of an object by using the class description for a template, querying the user to supply appropriate values for each of the varying attributes and writing out the object description in ODL.

PDS will create the initial object description libraries (and define the rules for accessing them), define an initial hierarchy of object classes, and create class templates for the initial object classes.

## I.2 ODL SPECIFICATION

### I.2.1 Definitions

The following terms are used in the label specification material.

*Attribute* – An important characteristic of an object that is included within the object's description within a label. Some attributes describe the physical characteristics of the data (the number of channels in a spectrum, for example) and others describe the contents of the data (the target of an image, for example).

*Comments* – Each line may optionally include a comment. The comment begins with a slash/asterisk pair (/\*) and terminates with the end of the line on which it appears. A comment can be terminated prior to the end of the line with “\*/” if it is necessary to embed a comment in an entry line. Comments may not be continued to another line. It is allowable to have lines that contain only comments (such lines begin with the comment indicator and the comment extends for the entire line). Blank lines are also allowed between any two statements within a label.

*Keyword* – The name of an attribute. Synonymous with “element name” as used in the PDS data dictionary.

*Label* – A group of statements in the PDS Object Description Language which conveys information about a data unit and the objects within it.

*Line* – An ASCII text string terminating with a carriage return, line feed sequence. Any line may contain a comment (see below).



*Object* – A data entity that is contained within a data unit and that is described in the data unit's label or in an ancillary label.

*Pointer* – A pointer is a type of ODL statement that specifies one of the following: the location of a subsidiary data object within the current data object; the name of another data unit that contains data objects described in the current label; or the name of a data unit containing an ancillary label that further describes the current object.

*Statement* – A complete sentence in the Object Description Language. A statement may be entirely contained on one line or it may require several lines.

*Value* – A numeric, literal or string constant that represents the value of the named attribute for a particular object.

### **I.2.2 Label Format**

A label consists of "statements" in the Object Description Language. The label begins with the first line of the file that contains it and ends with a line that contains only the word END. Any characters following the END but prior to the start of the data records are ignored. If the label record containing the END line is padded out to some fixed length, it is recommended that the ASCII space character be used for padding.

Within a label there are three types of ODL statements: object statements, attribute assignment statements and pointer statements. Object statements provide a shell around attribute assignment statements to indicate which data object the attributes are describing.

### **I.2.3 Object Statements**

An object statement has the following format:

```
OBJECT = name
      (one or more ODL statements)
END_OBJECT {= name}
```

The "name" within the object statement is used to identify a specific data object within the data unit. All the attribute assignment statements contained within an object statement describe this data object. Pointer statements within the object statement point to other objects that are constituents of this object or to other data units that contain the object or information about the object. Object statements can also contain other object statements, providing a hierarchical nesting of objects.

### **I.2.4 Attribute Assignment Statements**

An assignment contains the name of an attribute and the value of that attribute for the object which the assignment statement is describing. Assignment statements have the following format:

```
keyword = attribute-value
```

The layout of each assignment is essentially free-form: blanks and tabs are typically ignored. Specifically, blanks or tabs before a name, between the name and the assignment (=) symbol, between the assignment symbol and the value, or after the value are ignored.

Because different types of terminals and printers have different ways of treating tab characters, it is recommended that tabs not be used in a label; use the blank character instead.

#### I.2.4.1 Keyword

The keyword is equivalent to the element name in the PDS data dictionary. It is the formal PDS identifier for the data value. Keywords in the ODL are not limited to any set length but other PDS systems do have limitations on name lengths, so it is recommended that keywords not exceed 30 characters in length.

#### I.2.4.2 Assignment Symbol

The name and value must be separated by an "=" sign.

#### I.2.4.3 Value

The value field may contain a numeric, a literal, a text string, or a parenthesized list of values.

- (1.) *Numeric Values* – Numeric values can be signed or unsigned decimal or non-decimal integers or real numbers. Real number values must have an explicit decimal point in them and they may be represented in unscaled notation (like 3.14159) or in scaled (scientific) notation (like 31.4159E-1). There is no specific limitation on the magnitude of number that can be represented nor is there a limitation on the precision with which a floating point number can be represented; however, the computer on which the label will be processed typically imposes restrictions on magnitude and precision and users must be aware of these when building labels. A numeric value can optionally be followed by a units designator. The units value is enclosed in angle brackets.

Example of a numeric value followed by a units designator:

VELOCITY = 16.578 <KM/SEC>

Integer values can also be represented in bases other than base 10. Any number base from 2 (binary) to 16 (hexadecimal) is allowed.

- (2.) *Literals* – Literals are text fields following the same construction rules as names, or text fields enclosed in apostrophes if they contain special characters. Apostrophes may not be embedded in literals. Literals are used to indicate one value drawn from a finite (and usually rather small) set of possible values. For example, FILTER = BLUE could be used to indicate that the blue filter for a particular instrument, out of a set of BLUE, RED and GREEN filters, has been selected. It is recommended that literal names be kept to 30 characters or less.
- (3.) *Dates and Times* – Dates and time values can be represented explicitly in the ODL. The date can be given as either year, month and day of month or as year and day of year. Times are given as hours, minutes and seconds. There are provisions for specifying UTC time and for specifying a time in a time zone other than Greenwich.
- (4.) *Text Strings* – Strings may be any length and may consist of any sequence of printable ASCII characters, tabs or blanks enclosed in double quotes. Strings continue until a terminating double quote symbol is encountered. Double quote characters may not be embedded within quoted strings.

As noted, text strings are not limited in length and they often span two or more label lines. When the text string is read into the computer the text string is converted into one long string of characters according to the following rules:

- (a.) If the last character of a text string line is neither a "-" nor a "&" then one blank character will be placed between the last word of the line and the first word on the next line. If there are blank or tab characters after the last word of a line or before the first word of a line, they are ignored. This is the standard convention for written English text.

- (b.) If the last character of a line is a “-” (hyphen) character then the hyphen is removed and there will be no space between the last non-blank character on a line and the first non-blank character on the next line.
- (c.) If the last character of a line is an “&” (ampersand) then the ampersand is removed and the first characters of the next line will be placed immediately after the character before the ampersand, regardless of whether that character is blank or non-blank.
- (d.) You can explicitly indicate the end of a line of text by inserting a new-line indicator - \n - into a text string. For example, the ODL text string:

“This is the first line \n And this is the second.”

will result in the following:

This is the first line  
And this is the second.

- (5.) *Aggregate Values* – An attribute value may be either a scalar – a single value of one of the types described above – or an aggregate value which is built up from individual scalar values. There are two types of aggregate values allowed: arrays and sets. Arrays can be either one- or two-dimensional and all the elements of the array must be of the same data type (integer, real, date/time, literal or string). Arrays are specified in a way that makes the row and column orientation explicit and the elements of an array are always retained in the order in which they are given. A set is collection of zero, one or more values drawn from a superset of discrete values. Because they must be discrete values, only literals and integers are allowed as elements of a set (but you can’t mix literal and integer values within a single set). The null set is also allowed. The order in which the elements of a set are given is not important.

## I.2.5 Object Description Language Syntax Specification

In the following syntax specification, brackets (“[” and “]”) are used to indicate components that are optional: brackets by themselves mean that there can be either zero or one occurrence of the components that appear within the brackets; an asterisk (“\*”) immediately after the closing bracket means that zero, one or more occurrences of the components in the brackets may appear; and a plus sign (“+”) immediately after the closing bracket indicates that one or more occurrences of the components should appear. The vertical line (“|”) means ‘or’.

### I.2.5.1 Basic Elements of the Language

digit	:= 0-9
extended_digit	:= 0-9, A-F, a-f
letter	:= A-Z, a-z
character	:= any printing ASCII character plus tab
separator	:= space, tab, comma

### I.2.5.2 Lexical Elements

word	:= letter [letter   digit]*
name	:= word [_ word]* [_ unsigned_integer]
symbol	:= [character]+
sign	:= +   -
integer	:= [sign] [digit]+

```

unsigned_integer := [digit]+
extended_integer := [extended_digit]+

real             := unscaled_real | scaled_real
unscaled_real   := [sign] [digit]+ . [digit]* | [sign] . [digit]+
scaled_real     := unscaled_real E|e integer

date             := year_doy | year_month_day
year_doy        := year - doy
year_month_day  := year - month - day
year            := unsigned_integer
month           := unsigned_integer
day             := unsigned_integer
doy             := unsigned_integer

time             := local_time | utc_time | zoned_time
local_time      := hour_min_sec
utc_time        := hour_min_sec Z
zoned_time      := hour_min_sec sign hour
hour_min_sec    := hour : minute : second
hour            := unsigned_integer
minute          := unsigned_integer
second          := unsigned_integer [. unsigned_integer]

date_time       := date T time

```

### I.2.5.3 Syntactic Elements

```

label           := statement*
                END

statement       := object_stmt | attribute_stmt | pointer_stmt

object_stmt     := OBJECT = name
                statement*
                END_OBJECT [= name]

attribute_stmt  := keyword = attribute_value
keyword         := name
attribute_value := scalar_value | array_value | set_value | range_value
scalar_value    := integer_value | real_value | date_time_value |
                literal_value | text_string
array_value     := one_dim_array | two_dim_array
one_dim_array   := ( scalar_value [separator scalar_value]* )
two_dim_array   := ( one_dim_array
                    [one_dim_array]*
                    one_dim_array )
set_value       := { set_element [separator set_element]* } | {}
set_element     := literal_value | integer_value

```

```

integer_value    := integer | integer units_expression |
                  radix # extended_integer #
radix            := unsigned_integer
real_value       := real | real units_expression
range_value      := integer_value . . integer_value
date_time_value := date | time | date_time
literal_value    := name | ' symbol '
text_string      := " symbol "
units_expression := < units [units_operator units]* >
units            := name
units_operator   := * | / | ^

pointer_stmt     := ^ name = pointer_value
pointer_value    := file_name_value | position_value
file_name_value  := literal_value
position_value   := unsigned_integer

```

#### I.2.5.4 Semantics

This subsection discusses basic semantic aspects of the ODL as defined above.

- (1.) *Dates and Times* – The formats for dates and times in the ODL are a subset of the formats defined by the International Standards Organization recommendations on representations of dates and times as given in standard ISO/DIS 8601.

The year can be either a full specification of year Anno Domini (i.e., 1989) or it can be given modulo 100 (i.e., 89). If it is given in the later format, then it is interpreted to be a year in the current century. We strongly recommend that only full year specifications be used in labels. The month should be a number between 1 and 12. The day of month should be a number in the range 1 to 31, as appropriate for the particular month and year. The day-of-year should be in the range 1 to 365, or 366 in leap years.

Hours should be in the range 0 to 23. Minutes should be in the range 0 to 59. Seconds should be a number greater than or equal to 0.0 and less than 60.0.

- (2.) *Number Representations* – A radix for an integer value that is given in non-decimal format should be in the range 2 to 16. The most common radix values are 2 (binary), 8 (octal) and 16 (hexadecimal).

There is no defined maximum magnitude or precision for numbers specified in the ODL. In general, the actual range and precision of numbers that can be represented will be different for each kind of computer used to read or write a label. Developers of label reading/writing software will document the magnitude and precision of numbers that can be represented with their software on various computers. When label reading software cannot represent a number of the magnitude specified for a value in a label, then the software shall report that condition as an error to the user.

- (3.) *Array Representations* – All elements of an array must have the same value type (integer, real, date/time, literal or string.) Two-dimensional arrays are represented with a block structure that makes explicit the row-column relationships of the data; it is incumbent upon any software that processes a two-dimensional array to preserve this row-column relationship.
- (4.) *Set Representations* – The elements of a set must be either be all literals or all integers: intermixing the two is not permitted. There is no requirement to retain the order of set

elements when they are processed by label-reading software.

## Appendix J

### DATA UNIT FORMATS

The implications of file and record formats vary with the operating system being used. MS-DOS and UNIX systems do not have any special file or record formats and merely maintain a file length indicator in the directory. Basically, the files are considered to be a stream of bytes which the user's application program must interpret. Control of record structures is performed for text files by interpreting carriage return and/or line feed sequences as record delimiters. Operating systems such as VAX VMS provide dozens of unique file formats. It is recommended that file formats unique to a specific operating systems not be used for archival storage of PDS data products.

The basic keywords which identify file formats are RECORD\_TYPE and RECORD\_BYTES. The RECORD\_TYPE shall be either FIXED\_LENGTH, VARIABLE\_LENGTH or STREAM as shown in Figure J-1.

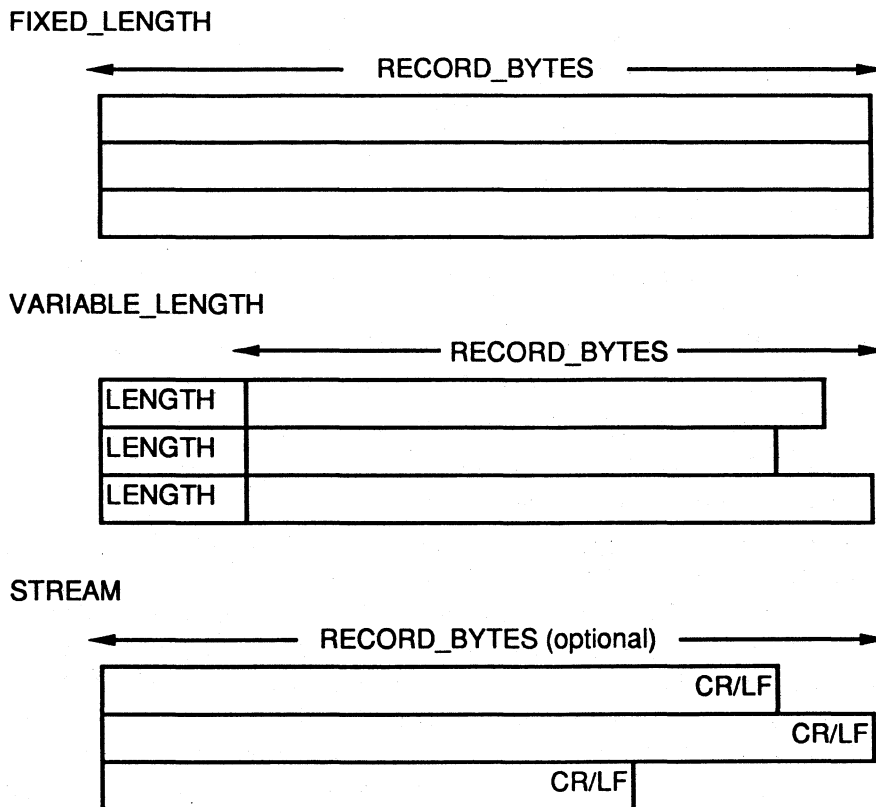


Figure J-1: Forms of Record Types

The RECORD\_BYTES parameter indicates the maximum number of 8-bit bytes in each record within the file. For FIXED\_LENGTH files the RECORD\_BYTES parameter is the record length. For VARIABLE\_LENGTH records the RECORD\_BYTES parameter is the maximum number of data bytes (excluding the 2-byte length indicator) in any record in the file. For STREAM records, the RECORD\_BYTES parameter is optional and, if specified, indicates the maximum length of any record, including the terminating carriage return/line feed characters.

## J.1 RECOMMENDATIONS FOR USING RECORD FORMATS

The choice of the proper record format is determined by the applications for/by which the data will be used. In general, fixed\_length records are well-suited to the storage of binary data files, such as images, binary tables or cubes, which are expected to be used or transported in structured environments. Variable\_length files are less transportable and require special software to read, and therefore are discouraged except for instances where they may optimize storage efficiency or access. An example of such an application is the compressed image format being used for CDROM storage. In this case the variable length structure allows "semi-random" access to any line in an image so that the entire file need not be decompressed whenever a portion of an image is needed. Stream formats should be used for text files and ASCII tables, to optimize storage efficiency and for ease of transportation to different computer architectures. The use of stream formats for binary data is discouraged. First, the stream record delimiters may occur as instances of valid data within a binary data file and second, large streams of binary data without delimiters can cause system buffers to overflow in record-oriented (VAX) systems.

RECORD TYPE	<i>fixed</i>	<i>variable</i>	<i>stream</i>
<i>Data form</i>	binary	binary	ascii
<i>Environment</i>	structured	very-structured	ad-hoc
<i>Volume</i>	large	very large	modest
<i>Media</i>	tape,disk	disk	electronic,others

### J.1.1 Fixed Length Record Formats

Fixed length record formats include two major categories, one where the fixed physical length maps directly to the logical length (that is, one physical record for each image line, or one physical record for each table record); and one where the fixed length is arbitrary, and provides only a physical unit of data for input/output operations, to facilitate processing the file.

The former approach is recommended, to make it easier to understand the file structure and to provide fairly simple file access with a variety of applications. In this approach, objects within a file are all stored in integral multiples of the basic logical data record length (RECORD\_BYTES).

In the latter case the record type might more appropriately be called UNDEFINED, however, in practice users generally call them FIXED\_LENGTH and use this length as a buffer size for input/output operations. The FITS format is an example of this application of the FIXED\_LENGTH record type. FITS files consist of fixed length records of 2880 bytes each (the lowest common denominator of all byte and word sizes on computer systems in use during the 1970's). The label records at the beginning of a FITS file indicate how the physical records are to be interpreted as data records. Within the FITS structure data records are packed into as many of these 2880 byte physical records as are needed. There is no filler added between logical records; the next logical record begins in the next byte of the current physical record throughout the file. A slightly different approach is taken in the USGS Flagstaff image file formats. All files are written as 512 byte physical records (the VAX internal record storage size) and logical records are packed into as many as are needed for a logical record; however, the remainder of that 512 byte record is not used, and the next logical record begins in the next 512 byte physical record. This allows for extremely fast access to image lines embedded in an image file, since a program can directly and immediately access the beginning of any logical record. In the FITS structure, additional computations would be required to find the correct physical record, translate that into a VAX block (blocks are 512 bytes per block) and position to the starting byte of the logical record within the buffer.



Both of these approaches require that the user perform somewhat complex buffer maintenance procedures to correlate logical and physical records.

### J.1.2 Variable Record Formats

A second category of record type is variable length. Each variable length record begins with a binary length field followed by the data. The variable length type is the default for files created with the VAX text editor. There are 2 commonly used forms, records with carriage control and without carriage control. Variable length records with carriage control begin with the 2 byte length field and are terminated by a HEX 0A (line feed). These records can not be used with binary data files. Variable length records without carriage control use can be used to store binary data.

PDS data files using variable length records should follow the CDROM and VAX VMS conventions where the records are preceded by a 2 byte (swapped or unswapped) integer which defines the length of the record with no carriage control.

NOTE: This is complicated slightly because the actual number of bytes following the length field is always an even number, thus all variable length records are an even number of bytes on the media. The reader software must physically read an extra pad byte if the length field is odd, unless the system software handles this (as on the VAX). Variable length records will be used for compressed data files, but their use in other situations is discouraged.

### J.1.3 Stream Record Formats

Stream records consist of ASCII text delimited with a carriage\_return and/or line\_feed sequence. The handling of these two ASCII codes on different computers and in communicating them between computers is quite different. On the other hand, stream files are the only type which can generally be transmitted safely on TEXT oriented communications facilities like TELEMAIL, NASAMAIL, or VAXMAIL. They are also usually editable with standard text editors. Fixed and variable length files pose many problems for ASCII text transmission and processing systems.

### J.1.4 Composite Files

Composite files utilize a STREAM format label file which points to the data file, which may be fixed, variable or stream. The rule for interpreting the label file is that it does not describe itself. All descriptive entries apply to the file(s) pointed to by the label file. Thus a RECORD\_TYPE = FIXED\_LENGTH keyword in a LABEL file refers to the record type of the data file, not to the LABEL file itself. Composite label files should always carry the file extension ".LBL" so that they may be identified by processing software.

The following examples show the major parameters for the three record types. Note the pointers (^object\_name = record\_number) to the starting location of objects. If the object is actually stored in another file, then the pointer will give the file name, not the record number (see the COMPOSITE FILE example).

#### FIXED LENGTH FILE

-----

RECORD_TYPE	=	FIXED_LENGTH
RECORD_BYTES	=	836

```

FILE_RECORDS           = 806
LABEL_RECORDS         = 3
/*           POINTERS TO STARTING RECORDS OF MAJOR OBJECTS IN FILE
^IMAGE_HISTOGRAM      = 4
^ENGINEERING_SUMMARY  = 6
^IMAGE                = 7

```

VARIABLE LENGTH FILE  
-----

```

RECORD_TYPE           = VARIABLE
RECORD_BYTES          = 836 /* interpret as max record bytes
FILE_RECORDS          = 806
LABEL_RECORDS         = 56
/*           POINTERS TO STARTING RECORDS OF MAJOR OBJECTS IN FILE
^IMAGE_HISTOGRAM      = 57
^ENCODING_HISTOGRAM   = 59
^ENGINEERING_SUMMARY  = 62
^IMAGE                = 63

```

STREAM FILE  
-----

```

RECORD_TYPE           = STREAM
FILE_RECORDS          = 806 /* if available, otherwise
                          /* determine based on parsing
                          /* the file.

```

COMPOSITE FILE  
-----

```

RECORD_TYPE           = FIXED_LENGTH
RECORD_BYTES          = 836
FILE_RECORDS          = 806
/*           POINTERS TO STARTING RECORDS OF MAJOR OBJECTS IN FILE
^IMAGE                = 'IMAGE.DAT'

```

## Appendix K

### DATA OBJECT DESCRIPTIONS

Data objects are categorized as elementary, aggregate or compound objects.

Elementary objects	data item types like INTEGER, REAL, CHARACTER, TIME, BIT_STRING, identified by ITEM_TYPE or TYPE key words.
Aggregate objects	the values of the homogeneous arrays, like HISTOGRAM and SPECTRUM, identified by presence of the ITEMS keyword in the object definition.
Compound objects	composed from elementary and aggregate objects, e. g., TABLE, IMAGE, QUBE, TEXT, HISTORY

#### K.1 ELEMENTARY OBJECTS

The data type definition segments of the ODL label structure use a simple syntax to define the data types of stored data values. The data format specifications indicate to the parser the type of data in a field, the starting byte location and the length in bytes. An optional format specification can be given for use in displaying data item values. *Within all the object descriptions, the byte, bit and record POSITIONS count from 1, and from left to right where applicable.*

Data values may be represented within data files in ASCII or BINARY format. The ASCII storage format is much simpler to transfer between different hardware systems and often between different application programs on the same computer.

On the other hand, all numerics are stored and manipulated internally in binary numeric types; thus ASCII data values must be converted to internal formats before they can be processed. In addition, the ASCII representation of most numeric values requires more storage space than does the binary format. For example, each 8 bit pixel value in an image file would require 3 bytes if stored in ASCII format.

The current specification uses the same set of data types for both ASCII and binary data values. The basic interpretation of whether a data value is stored in ASCII or BINARY format within the data record is derived from the object type. For TEXT and TABLE objects the default representation is that the data type is stored in an ASCII format. For the IMAGE, CUBE and QUBE object types the default representation is that the data value is stored in binary format.

The following data types can be specified:

INTEGER	A signed integer value [default = 4 bytes].
REAL	A real (floating point) value [default = 4 bytes].
BIT_STRING	Used to represent information defined at the bit level.
COMPLEX	A complex value. Usage not currently defined.

CHARACTER (or STRING)

A text string.

TIME

Time value. Only the ASCII format is currently defined.

### K.1.1 Specification Qualifiers

Additional qualifiers can be used to explicitly state the length or representation format for binary values.

#### K.1.1.1 Length Specification

The interpretation of binary fields is normally determined by the field length (BYTES or BITS) parameter, thus an integer field specified with a bytes value of 2 would be interpreted as a 16-bit signed binary value. The length can also be specified by appending a byte count to the end of the type specification (-1, -2, -4 or -8).

INTEGER\_1

A signed 1 byte integer value.

INTEGER\_2

A signed 2 byte integer value.

#### K.1.1.2 Binary Integer Specifications

There are two widely used formats for integer representations in 16-bit and 32-bit binary fields. These are the most-significant-byte first (MSB) and least-significant-byte first (LSB) architectures. The MSB architectures are used on IBM main-frames, many UNIX minicomputers (SUN, Apollo) and Macintosh computers. The LSB architectures are used on VAX systems and IBM PCs. The default interpretation for ODL files is the MSB architecture. Therefore files written on VAX or IBM PC hosts should specify LSB\_INTEGER for the field type of binary integers, or use synonyms for LSB, which are VAX and IBMPC. Alternatively the structure definition can be initiated with a HARDWARE\_TYPE keyword indicating MSB or LSB is to be used for all fields in the structure.

LSB\_INTEGER

A signed 4-byte integer value in least-significant-byte-first order.

VAX\_INTEGER\_2

A signed 2-byte integer value in least-significant-byte-order.

#### K.1.1.3 Signed Versus Unsigned

The default binary integer is a signed value in 2's complement notation. Alternatively, the TYPE field may have the term "UNSIGNED\_" prepended to the specification.

UNSIGNED\_INTEGER\_1

An unsigned 1 byte integer value.

UNSIGNED\_INTEGER\_2

An unsigned 2 byte integer value.

#### K.1.1.4 Floating Point Formats

The standard representation for floating point numbers is as defined in ANSI/IEEE 754. Floating point values which use other representations should preface the type parameter with an identification of the host machine. In the case of VAX double precision floating point, two representations are used, the standard DOUBLE and a "G" format. The "G" format will be specified as VAXG\_DOUBLE. The hardware specification can be set for an entire object definition by using the HARDWARE\_TYPE keyword in the STRUCTURE definition.

VAX\_REAL

A real (floating point) value [default = 4 bytes] in VAX format.

VAX\_DOUBLE (or REAL\_8)

An 8 byte double precision floating point value. Doubles should be used if the precision required for a numeric value exceeds 7 digits.

VAXG\_DOUBLE

A special type to handle the VAX G-type double precision.

#### K.1.1.5 Bit String Data

A BIT\_STRING is used as a container to hold and process individual bit item values. A BIT\_STRING can be a 1-byte, 2-byte or 4-byte field, much like a binary INTEGER. Extraction of BIT\_STRING data for 2-byte and 4-byte fields is dependent on the host architecture, and follows the specifications of the integer representation supplied above. In processing bit values within a BIT\_STRING, any necessary conversions (from MSB first to LSB first, for example) should be done before extracting the individual bit items. This will assure that bit fields are not fragmented due to differences in the hardware architecture. The default bit string is MSB first. Therefore files written on VAX or IBM PC hosts should specify LSB\_BIT\_STRING for the field type of binary integers. Alternatively the structure definition can be initiated with a HARDWARE\_TYPE keyword indicating MSB or LSB is to be used for all fields in the structure.

Data objects defined within a BIT\_STRING must be of the type INTEGER or UNSIGNED\_INTEGER. Bit locations are assigned counting from left to right, starting with bit 1.

BIT\_STRING\_2

Specifies an item of 16-bits (equivalent to an UNSIGNED\_INTEGER\_2) containing sub-items defined at the bit level.

LSB\_BIT\_STRING

Specifies an item of 32-bits (equivalent to an LSB\_UNSIGNED\_INTEGER) containing sub-items defined at the bit level.

#### K.1.2 Object Format Specifications

This discussion is a great simplification of the data format specification question. It only addresses fundamental types. The data format specification is used to determine the format for display of a data value. A 4 byte binary integer can store values in the range of -2,147,483,648 to 2,147,483,647 however the actual values stored in the field may only range from -9999 to 9999. In this case it is convenient to specify the output length with a format statement "I5" where I indicates that the value is an integer and 5 indicates the number of display positions (one for the sign and 4 for the numeric value).

The following FORTRAN data format specifications will be used:

Aw	Character (alphanumeric) data value.
Iw	Integer value.
Fw.d	Floating point value, displayed in decimal format.
Ew.d[Ee]	Floating point value, displayed in exponential format.

where:

w = Total number of positions in the output field (including sign, decimal point or "E").  
d = Number of positions to the right of the decimal point.  
e = Number of positions in exponent length field.

### K.1.3 Explicit Definitions of Elementary Objects

Where the non-standard data types described above cannot be used, special data type specifications may be defined. These definitions shall be specified as in the General Data Interchange BINREP format (Section 5.3.3.3, JPL Document D3606, F. Billingsley, 1988-01-12), but cast in the ODL object structure definition format.

The keywords to use in describing these non-standard data types are:

COMPLEMENT	Representing 0, 1 or 2's complement.
EXCESS	in decimal integer form.
BASE	which is raised to the exponent - excess power.
SIGN	bit position of sign.
EXPONENT	bit positions of exponent.
MANTISSA	bit positions of mantissa.
EXPONENT_SIGN	bit position of sign exponent.
ORDER	bit positions of bits making up integer value field.
INVERT	bit positions in which the "1s" and "0s" are inverted.
IMPLICIT	flag indicating an implicit 1 bit in the mantissa.

The numerical value represented is:

(Sign)Mantissa \* Base \*\* Exponent\_Sign(Exponent - Excess)

The values in sign, exponent, mantissa, exponent\_sign, and order are ASCII numbers indicating the bits in which the parameter occurs, msb first, separated with commas. The leftmost bit is designated bit 1. The notation m..n may be used to designate the range of bit positions m to n, inclusive.

Excess is the excess (base 10) coding of the exponent.

The examples below show the definition of a VAX INTEGER and DOUBLE using this notation.

```
OBJECT          = VAX_INTEGER
TYPE            = TEMPLATE
COMPLEMENT      = 2
SIGN            = 25
ORDER           = (26..32,17..24,9..16,1..8)
END_OBJECT
```

```
OBJECT          = VAX_REAL
TYPE            = TEMPLATE
COMPLEMENT      = 2
BASE            = 2
EXCESS          = 128
SIGN            = 9
IMPLICIT        = TRUE
MANTISSA        = (2..8,25..32,17..24)
EXPONENT        = (10..16,1)
```

NOTE = "Byte values AB CD EF GH are converted to  
CD AB GH EF, where there is a one-bit sign  
followed by an 8 bit exponent,  
followed by a 23 bit mantissa."

END\_OBJECT

## K.2 AGGREGATE OBJECTS

Aggregate objects consist of homogeneous arrays of elementary objects. Aggregate objects are identified by the ITEMS keyword in the object definition.

The following example illustrates a HISTOGRAM object:

```
OBJECT      = IMAGE_HISTOGRAM
ITEMS       = 256
ITEM_BITS   = 32
ITEM_TYPE   = INTEGER
END_OBJECT
```

## K.3 COMPOUND OBJECTS

Compound objects represent collections of elementary and aggregate objects. Five major compound object types have been identified:

TEXT	a format for documents
TABLE	a flat file of ASCII and/or binary values
IMAGE	a special array for images (special case of QUBE)
QUBE	an array for multi-dimensional data files with optional prefix and suffix data in each dimension
HISTORY	a format for cumulative ASCII keyword processing histories

### K.3.1 TEXT Object Format

The default text object consists of ASCII text in STREAM format with each line separated by a carriage return/line feed pair. Lines should be 71 characters or less, and there should be no embedded control characters other than the form feed (control-L) and tab characters (Control-I).

Example of a TEXT object:

```
NJPL1I100PDS100000000 = SFDU_LABEL
RECORD_TYPE = STREAM
OBJECT      = TEXT
END_OBJECT
END
Now is the time for all good men to come to the aid of their
labels.
```

### K.3.2 TABLE Object Format

A PDS TABLE object is a uniform collection of records containing ASCII and/or binary value fields, as described in the label. If all of the fields are ASCII, the table has `FORMAT = ASCII`, and it may have stream (the default) or fixed-length records. Otherwise it has `FORMAT = BINARY` and it **MUST** have fixed-length records.

If a table has ASCII format and stream records, the records are terminated with carriage return / line feed pairs. The most widely used form has fields separated by commas, and text items enclosed in double quotes. This kind of table can be processed by nearly all data management systems. It is recommended for PDS tables which are expected to be used with commercial data management or data analysis software.

The basic file structure is defined by the following keywords:

**FORMAT** The type or representation of data stored in the table. The default table format is ASCII, indicating that values are stored as ASCII text. Tables may also have `FORMAT = BINARY` where data items are stored in binary (or mixed binary and ASCII) format.

**RECORD\_TYPE** The default `RECORD_TYPE` of an ASCII table is `STREAM`, while that of a `BINARY` table is `FIXED_LENGTH`. The parameters which define a `BINARY TABLE` are identical to those for an `ASCII TABLE`, except the default meaning of the column type refers to a binary storage format. For example, `INTEGER` column type in a `BINARY TABLE` would indicate a signed 4 byte (32 bit) binary value.

**FILE\_RECORDS** The number of physical records.

Keywords used to define table contents are as follows:

**TABLE\_RECORD\_BYTES** The number of bytes in a fixed-length table record.

**ROWS** The number of logical table entries.

**TABLE\_RECORD\_ROWS** The number of table rows contained in a single table record.

**ROW\_BYTES** The number of bytes in each table row.

**ROW\_COLUMNS** The number of items of information in each table row.

Keywords used to define the data objects (columns) within the table are:

**NAME** The name of a data item or column in a table row.

**TYPE** The data type of the data item, `INTEGER`, `REAL`, `DOUBLE`, `CHARACTER`, `STRING`, `TIME`, `BIT_STRING`. See Appendix K for a description of valid data types.



FORMAT

A Fortran representation of the format statement needed to read or write the data item. See Appendix K for a description of valid formats

START-BYTE or START-BIT

The byte or bit position (counting from 1) of the beginning of the data item within the row.

BYTES or BITS

The number of bytes or bits containing the data item.

BYTE or BIT

The byte or bit position (counting from 1) of a 1-byte or 1-bit data item within a row. BYTE may substitute for the 2 statements START-BYTE = n and BYTES = 1. BIT may substitute for START-BIT = n and BITS = 1.

UNIT

The units of measure of the data item.

NOTE

Descriptive notes about the data item.

Example of an ASCII TABLE file:

```
NJPL1I10PDS100000000    = SFDU_LABEL
RECORD_TYPE              = STREAM
OBJECT                   = TABLE
  FORMAT                 = ASCII
  ROWS                   = 3
  ROW_COLUMNS            = 3
  OBJECT                 = COLUMN_1
  TYPE                   = INTEGER
  END_OBJECT
  OBJECT                 = COLUMN_2
  TYPE                   = REAL
  END_OBJECT
  OBJECT                 = COLUMN_3
  TYPE                   = CHARACTER
  END_OBJECT
END_OBJECT
END
11111, 22.22, "ABCDE"
22222, 33.33, "FGHIJ"
33333, 44.44, "KLMNO"
```

Example of a BINARY TABLE file:

```
NJPL1I10PDS100000000    = SFDU_LABEL
RECORD_TYPE              = FIXED_LENGTH
RECORD_BYTES             = 10
^TABLE                   = 'TABLE.DAT'
OBJECT                   = TABLE
  FORMAT                 = BINARY
  ROWS                   = 3
```

```

ROW_COLUMNS           = 3
OBJECT                = COLUMN_1
  TYPE                = REAL
  START_BYTE          = 1
  BYTES               = 4
END_OBJECT
OBJECT                = COLUMN_2
  TYPE                = INTEGER_2
  START_BYTE          = 5
  BYTES               = 2
END_OBJECT
OBJECT                = COLUMN_3
  TYPE                = INTEGER
  START_BYTE          = 7
  BYTES               = 4
END_OBJECT
END_OBJECT
END

```

### K.3.3 IMAGE Object Format

The image object format is designed for simple two-dimensional arrays from imaging type instruments (cameras, radar, etc.). For fixed format image files there may be a label group, a header object, a history object, the image object, and a trailer object, each of which will require a separate definition. In addition, it is common for the data records to have either prefix or suffix bytes with each record of data, representing time tags, line numbers or engineering parameters specific to a certain line of data.

The physical and logical structure of any of these files can be defined with the following label parameters:

#### RECORD.TYPE

Image objects may have `FIXED_LENGTH`, `VARIABLE_LENGTH` or `STREAM` records. `FIXED_LENGTH` is the default. Images in `STREAM` format would have to have ASCII item types.

#### RECORD.BYTES

The record length parameter represents the physical length of each record in the file. It also represents the `DEFAULT` logical record length for components of the file.

#### FILE.RECORDS

The file records parameter represents the number of physical records within the file with each record having a length equal to the `RECORD.BYTES` value.

#### LABEL.RECORDS

Number of records containing PDS text labels. Generally the label area will be filled so that the labels consume a multiple of the `RECORD.BYTES` parameter.

**LABEL\_RECORD\_BYTES**

Length of each label record. This parameter defaults to the RECORD\_BYTES if no value is provided.

These parameters provide a framework within which the logical file structure is built. The most common situation is that the record components have logical length values which are equal to the physical values, HOWEVER, the physical length values can be overridden for any component of an image file by specifying a "component\_RECORD\_BYTES" parameter.

The data format keywords for the IMAGE object are:

**IMAGE\_RECORDS**

Number of records containing image data.

**IMAGE\_RECORD\_BYTES**

Length of each line record. This parameter defaults to the RECORD\_BYTES if no value is provided. This value must be an integral number of 8-bit bytes.

**LINES**

Number of lines in image. Normally equal to IMAGE\_RECORDS value.

**LINE\_PREFIX\_BYTES**

Number of bytes of data which precede the image data in each line record.

**LINE\_SAMPLES**

Number of sample values contained in each line record.

**SAMPLE\_BITS**

Number of bits of data comprising one sample value. Common values are 1 (bit), 4 (nibble), 8 (byte), 16 (halfword), 32 (fullword).

**SAMPLE\_TYPE**

Data type of sample values, where the default is unsigned\_integer. Other data types are defined in Appendix K.1.

**SAMPLE\_BIT\_MASK**

A bit pattern representation indicating which bits are active in the SAMPLE value. For example "SAMPLE\_BIT\_MASK = 2#00011111#" indicates that only the lower 5 bits of each sample value contain valid data.

**LINE\_SUFFIX\_BYTES**

Number of bytes of data which follow the last sample value in a line record.

**TRAILER\_RECORDS**

Number of records which follow the last image line record in a file.

**TRAILER\_RECORD\_BYTES**

Length of each trailer record. This parameter defaults to the RECORD\_BYTES if no value is provided.

### K.3.4 QUBE Object Format

The qube object type is a generalization of the image object type to data objects with an unlimited number of dimensions. It is distinguished from an ordinary multi-dimensional array by its capacity for arbitrary prefix and suffix data in each axis. (See terminology below.) The image object type is a special case of the qube object type.

The primary motivation for the qube object is the spectral image cube generated by the imaging spectrometer class of instruments, a class with a growing number of members. For these cubes, the principal use of the prefix and suffix capability will be to include "backplanes" of associated data (geometry parameters, quality codes, etc.) in the qube. It may also be used for engineering data accompanying image lines.

A typical file of such data will contain a PDS label, a history object, optional header and/or other objects, and a qube object. The label will contain descriptions of the structure of the most important parts of the file, and pointers to supplementary labels containing details about the less-often-used parts.

#### K.3.4.1 Qube Terminology

Note the distinction between qube, cube and the various image cubes. Because of its frequent occurrence, the image object type is identified as a special case of the qube object type. See Figure K-1 for a diagram of an "Image Cube".

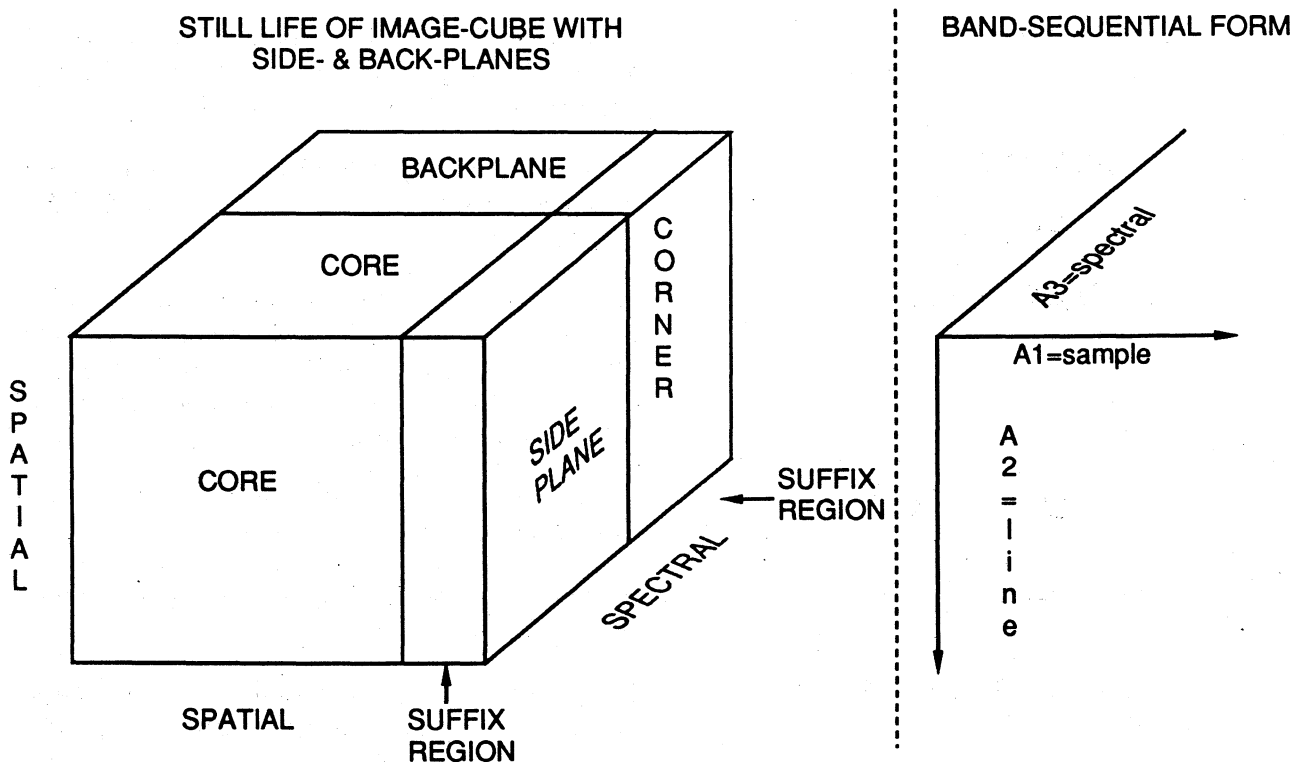


Figure K-1: Diagram of an "Image Cube"

Qube	A multi-dimensional array with optional prefix and suffix areas in each dimension. (final name still open).
Cube	A qube of 3 dimensions
Image cube	A cube of multiple images
Spectral image cube	An image cube one of whose dimensions is wavelength (band)
Image	A qube of 2 spatial dimensions: sample and line
Item	An elementary object (data element) in the multi-dimensional array, or in its prefix or suffix regions.
Sample	synonym for item in an image
Axis	A dimension of a qube
Core	The "central" region of a qube corresponding to the principal type of data in all axes. The core is "homogeneous", i.e. all elements are of the same type. A qube with prefix or suffix elements will be "heterogeneous".
Prefix	Item(s) preceding the core along an axis of a qube (also prefix region, prefix plane, etc.)
Suffix	Item(s) following the core along an axis of a qube (also suffix region, suffix plane, etc.)
Backplane	A plane of a cube perpendicular to the 3rd dimension with index greater than the highest index of any plane in the core
Sideplane	A plane of a cube perpendicular to either the 1st or 2nd dimension with index outside the index-range of the core
Corner	Region of a qube corresponding to prefix/suffix indices in two or more axes. Corner regions are normally left empty.
Co-cube	Backplanes or sideplanes of a cube in a separate file (these generally have no core; planes are of different items on the same space)
Bin	A 1 to N dimensional region in the "true physical space" with finite extent along one or more axes represented by an element in the core of a qube.

### **K.3.4.2 Label Keywords Describing the File Containing a Qube Object**

**RECORD\_TYPE** [literal]

**FIXED\_LENGTH**

**RECORD\_BYTES** [integer]

record length in bytes

**FILE\_RECORDS** [integer]

total number of records in file

**LABEL\_RECORDS** [integer]

number of records in label

### **K.3.4.3 Basic Qube Object Keywords**

**TYPE** [literal]

**QUBE** – multidimensional array with optional prefix/suffix data in all dimensions

**IMAGE** – synonym for 2-d qube of sample and line dimensions

**AXES** [integer]

number of dimensions in the cube

**STORAGE\_TYPE** [literal]

name describing the sequence of items in the qube (the following are for image cubes).

**BAND\_SEQUENTIAL (BSQ)**

**BAND\_INTERLEAVED\_BY\_LINE (BIL)**

**BAND\_INTERLEAVED\_BY\_PIXEL (BIP)**

plus various tiled and compact forms (TBD).

**ITEM\_TYPE** [literal]

interpretation of data item (must be compatible with **ITEM\_BITS** below)

applies to all items in core and is default for prefix/suffix (See Appendix K.1)

**INTEGER, UNSIGNED\_INTEGER, REAL, COMPLEX, CHAR[ACTER], BIT\_STRING, ASCII\_INTEGER, ASCII\_REAL, ...** (default: **INTEGER**) (see **DATA TYPE SPECIFICATIONS** above)

**ITEM\_BITS** [integer]

item length in bits (usually  $n*8$ )

applies to all items in core and is default for prefix/suffix

### **K.3.4.4 Qube Axis Keywords**

**AXIS\_NAME** [vector of literals]

name of each axis e.g. (**SAMPLE, LINE, BAND**) [applies to core]

**AXIS\_UNIT** [vector of literals]

physical units of each axis [applies to core]

**AXIS\_NOTE** [vector of text]

descriptions lengthier than NAME & UNIT

AXIS\_ITEMS [vector of integers]

length of each axis, e.g. (128,128,83)

AXIS\_BYTES if not integral number of items

PREFIX\_ITEMS [vector of integers]

prefix length of each axis (default 0)

PREFIX\_BYTES in not integral number of items

CORE\_ITEMS [vector of integers]

length of each axis within core (defaults to AXIS\_ITEMS - PREFIX\_ITEMS - SUFFIX\_ITEMS)

SUFFIX\_ITEMS [vector of integers]

suffix length of each axis (default 0)

SUFFIX\_BYTES if not integral number of items

[The rest of the keywords in this section apply to the core only]

For uniform axes representing physical variables (e.g. wavelength):

AXIS\_INTERVAL [vector of reals]

increment between successive bins or grid points along each axis (1. is the default, 0. indicates a discrete set)

and choice of one of the following pairs of keywords:

AXIS\_START [vector of reals]

outer edge of first bin of core (def 0)

AXIS\_STOP [vector of reals]

outer edge of last bin of core (def 0)

AXIS\_FIRST [vector of reals]

central value of first bin of core

AXIS\_LAST [vector of reals]

central value of last bin of core

For non-uniform axes or spatially undersampled data (i is axis number):

AXIS<sub>i</sub>.BIN\_RANGE [vector of pairs of reals]

range along axis for each bin

or

AXIS<sub>i</sub>.BIN\_START [vector of reals]

starting value (along axis) for each bin  
and  
AXISi\_BIN\_STOP [vector of reals]  
stopping edge (along axis) for each bin

or  
AXISi\_BIN\_CENTER [vector of reals]  
central value for each bin  
and  
AXISi\_BIN\_WIDTH [vector of reals]  
width for each bin

#### K.3.4.5 Qube Core Keywords

CORE\_NAME [literal]  
name of principal data item (e.g. Brightness)  
CORE\_UNIT [literal]  
physical units of principal data item (default: dimensionless)  
CORE\_NOTE [text]  
description lengthier than NAME and UNIT  
CORE\_BIT\_MASK [binary integer]  
e.g. 2#01111111# for 7 bit item in byte

The following 3 keywords describe the relationship between the data stored in the core and the "true" physical values they represent. Only one of CORE\_MULTIPLIER and CORE\_DIVISOR may be present. All 3 keywords have default values.

`'true' value = base + (stored value * multiplier)`

or

`'true' value = base + (stored value / divisor)`

CORE\_BASE [real]  
base value to be added to stored value (default 0.)  
CORE\_MULTIPLIER [real]  
multiplier of stored value (default 1.)  
CORE\_DIVISOR [real]  
divisor of stored value (default 1.)

CORE\_EXCLUDE\_VALUE [type specified by ITEM\_TYPE]  
value denoting missing or null or "fill" or bad data (default is minimum numeric value allowed by hardware for type and length of item)



CORE\_EXCLUDE\_RANGE [vector of 2 values of type specified by ITEM\_TYPE]

algebraic minimum and maximum of range of values to be excluded from consideration as "valid" data (may be reserved for flagging various kinds of invalidity, to be described more fully in text)

The following four keywords are optional items – possibly vectors of values, one for each band in core – perhaps maintained in the cube header.

CORE\_MINIMUM [type indicated by ITEM\_TYPE]

CORE\_MAXIMUM [type indicated by ITEM\_TYPE]

CORE\_MEAN [type indicated by ITEM\_TYPE]

CORE\_STANDARD\_DEVIATION [type indicated by ITEM\_TYPE]

#### K.3.4.6 Qube Prefix/Suffix Keywords

In the following definitions, i = axis number, and j = "PRE" or "SUF".

AXIS<sub>i</sub>\_jFIX\_NAME [vector of literals]

= name of each prefix/suffix item, e.g. AXIS\_3-SUFFIX\_NAME = (LATITUDE, LONGITUDE, PHASE\_ANGLE)

and similar vector-valued keywords corresponding to Core keywords: UNIT, NOTE, ITEM\_TYPE, BIT\_MASK, BASE, MULTIPLIER, DIVISOR, MINIMUM, MAXIMUM, MEAN, STANDARD\_DEVIATION, EXCLUDE\_VALUE.

#### K.3.4.7 Projection Keywords

The following keywords apply to the image plane of qube.

PROJECTION\_NAME [literal]

cartographic projection

NULL

SIMPLE\_CYLINDRICAL

MERCATOR

LAMBERT\_CONFORMAL

POLAR\_STEREOGRAPHIC

SINUSOIDAL\_EQUAL\_AREA

POINT\_PERSPECTIVE (view)

etc.

PROJECTION\_NOTE [text]

PROJECTION\_ORIGIN [vector of 2 reals]

origin of map (lat/lon)

PROJECTION\_ORIGIN\_LOCATION [vector of 2 reals]

first line and sample relative to origin

PROJECTION\_RESOLUTION [unitized real]

resolution of map, e.g. deg/pixel or km/pixel  
 PROJECTION\_ELLIPTICITY [real]  
 ellipticity of object  
 POSITIVE\_LONGITUDE [literal]  
 EAST, WEST  
 STANDARD\_PARALLELS [vector of reals]  
 CENTER\_LONGITUDE [real]  
 CENTER\_LATITUDE [real]  
 LONGITUDE\_RANGE [vector of reals]  
 LATITUDE\_RANGE [vector of reals]

### K.3.5 History Object Format

A history object is a collection of text describing the processing performed to generate a data object. It consists of a series of history entries, one for each process the data has been subject to, followed by an END statement.

#### K.3.5.1 History Entry

Each history entry is a sub-object of the history object. The entry consists of a series of keyword=value statements, each terminated by CR/LF, beginning with OBJECT and ending with END\_OBJECT. One group of statements, the process parameters, form a nested object within the entry. All statements follow the syntax of the Object Description Language (ODL) used in the data unit label. For example:

	[comments, * = statement required]
OBJECT = FILT8B	[*value is process name]
VERSION_DATE = 1986-08-15	[*program version date]
VERSION_NUMBER = 2.3	[version number]
OBJECT = PARAMETER_GROUP (or PARAMETERS)	[*program parameters: names not necessarily in PDS dictionary; values might be homogeneous vectors.]
FILTER = NPE	
LINE = 256	
SAMPLE = 256	
NULL = 0	
FRACTION = .7	
FROM = input_file	
TO = output_file	
END_OBJECT	[*]
DATE_TIME = 1988-08-08T08:08:08	[*run date & time]
NODE = GRUMPY	[*(net) name of computer]
USER = "Snow White"	[*username]
PROGRAM_NOTE = "High pass filter"	[*program-generated description]
USER_NOTE = "Eliminate noise from Ganymede global mosaic"	[*user input]

PREDECESSOR = a (or {a1,a2,...})

[\*pointer(s) to predecessor entry(s) (record number(s))]

SUCCESSOR = b

[\*pointer to successor entry, to be added when that entry is created]

END\_OBJECT

[\*]

### K.3.5.2 Tree of Processes

The entire History Object has a kind of tree structure, with the file containing it as the root of the tree, and the branches extending backwards in time, rather like human ancestor trees, but where there can be one, two or more parents! There are branch points at each file merge step (i.e. when two or more input files combine to form one output file in some process.) The entries are therefor doubly linked by pointer statements for computer traceability. (See sample entry above.) The FROM and TO parameters, if used, also provide traceability, but it is indirect, and the parameter names may vary from program to program. In addition, an indentation technique is employed for human readability. A sample diagram of processing history is given in Figure K-2. A sample indented list is given in Figure K-3.

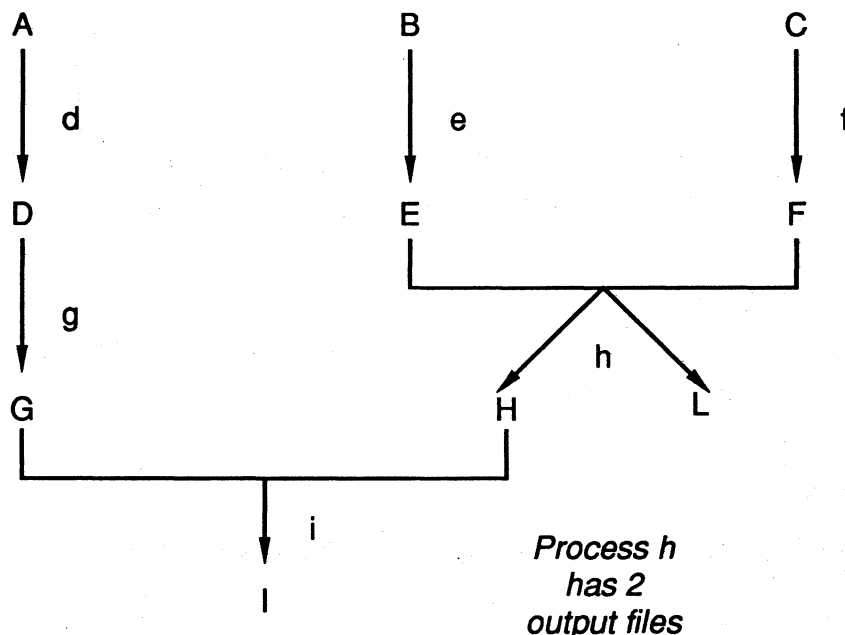


Figure K-2: Diagram of Processing History for a File

### K.3.6 History Object Processing

The following rules should be applied to help make the history object more human-readable:

- (1.) At each non-merge step, insert a blank line before adding the new entry. Do not indent.
- (2.) At each merge step, insert 2 blank lines, and indent merged entries by 1 space.
- (3.) A restriction of 75 characters per line will allow histories of up to 5 levels to be expressed in indented form with a maximum line length of 80 characters. (The indentation could be suppressed if there were more than 5 levels.)

Each application program should invoke history routines (to be provided) to:

**Only OBJECT  
and  
END\_OBJECT  
statements are  
shown here**

```
OBJECT = d  
. . . history of file D  
END_OBJECT
```

**History of File G**

```
OBJECT = g  
. . . history of file G  
END_OBJECT
```

```
OBJECT = e  
. . . history of file E  
END_OBJECT
```

```
OBJECT = f  
. . . history of file F History of File H  
END_OBJECT
```

```
OBJECT = h  
. . . history of file H  
END_OBJECT
```

```
OBJECT = i  
. . . last (current) entry in history of file I  
END_OBJECT
```

END

**Figure K-3: Textual Description of Processing History for a File**

- (1.) Update history objects in memory by adding a new history entry OR
- (2.) Copy history object of input file to output file and append new history entry OR
- (3.) Merge history objects of multiple input files (as in II above) and write to output file, appending new history entry

Each program should also:

- (1.) Supply a descriptive "program\_note" to be included in the entry.
- (2.) Provide for user input of "user\_notes" to be included in the entry.
- (3.) Maintain a version date to be included in the entry.
- (4.) Optionally maintain a version number according to standards to be adopted.

## Appendix L

### SAMPLE ODL LABELS

The following examples present ODL label sets for a variety of different data object types. They represent a working prototype implementation of the ODL label structure.

#### L.1 SAMPLE TEXT FILE LABEL

This label set precedes the description of the Voyager image documentation contained on the Uranus Image Disk.

```
NJPL1I00PDS100000000    = SFDU_LABEL
RECORD_TYPE              = STREAM
OBJECT                   = TEXT
FORMAT                   = TEX
END_OBJECT
SPACECRAFT_NAME          = VOYAGER_2
MISSION_PHASE            = URANUS_ENCOUNTER
INSTRUMENT_NAME         = {NARROW_ANGLE_CAMERA,WIDE_ANGLE_CAMERA}
END
```

## L.2 SAMPLE TABLE FILE LABELS

### Delimited ASCII Table of Spectral Reflectance Values:

Note that start\_byte and bytes parameters are not needed since all values are delimited with commas and each row comprises one record.

```
NJPL1I00PDS100006664      = SFDU_LABEL
RECORD_TYPE                = STREAM
OBJECT                     = TABLE
  FORMAT                   = ASCII
  ROWS                      = 125
  ROW_COLUMNS              = 3
  OBJECT                   = WAVELENGTH
  TYPE                     = REAL
  END_OBJECT
  OBJECT                   = REFLECTANCE
  TYPE                     = REAL
  END_OBJECT
  OBJECT                   = ERROR
  TYPE                     = REAL
  END_OBJECT
FILE_NOTE                  = "Data is taken from 'The Galilean
Satellites:  New Near-Infrared Spectral Reflectance (0.65-2.5
microns) and a .325-.5 micron Summary', Clark and McCord; Icarus,
vol. 41, 323-329 (1980).  Figure 13, Ganymede Leading. The
reflectivity is the geometric albedo scaled to 1.0 at 1.02
microns."
END
.350000, .488778, .022660
.375000, .590744, .011475
.400000, .632155, .012826
.433000, .752500, .011015
.466000, .783059, .008976
.500000, .869610, .007633
.533000, .918507, .004766
.566000, .955278, .001062
.600000, .973874, .005927
.633000, 1.010833, .006079
.666000, 1.021557, .009361
```

AND SO ON...

## PWS Binary Data File:

Note that there are many files with the same record structure so the structure definition is stored in a single STRUCTURE file and referenced in the labels for the individual data files.

*Labels for data file PWS072.TAB:*

```
NJPL1I0OPDS101033180      = SFDU_LABEL
RECORD_TYPE                = FIXED_LENGTH
RECORD_BYTES               = 48
FILE_RECORDS               = 21525
LABEL_RECORDS              = 32
^TABLE                     = 33 /* Location of start of table in records.
SPACECRAFT_NAME            = VOYAGER_1
MISSION_PHASE               = JUPITER_ENCOUNTER
TARGET_NAME                = JUPITER_MAGNETOSPHERE
INSTRUMENT_NAME            = PLASMA_WAVE_SPECTROMETER
INSTRUMENT_MODE            = SPECTRUM_ANALYZER
SPACECRAFT_EVENT_TIME      = 1979-072T00:00:00Z
OBJECT                     = TABLE
  FORMAT                   = BINARY
  ROWS                     = 21493
  ^STRUCTURE                = 'VGRPWSEL.FMT'
END_OBJECT
END
```

*Contents of referenced structure file VGRPWSEL.FMT:*

```
NJPL1I0OPDS100000000      = SFDU_LABEL
RECORD_TYPE                = STREAM
/* STRUCTURE TABLE FOR VGRPWS ELECTRIC WAVEFORM DATA
OBJECT                     = YEAR
  TYPE                     = VAX_INTEGER
  START_BYTE               = 1
  BYTES                    = 2
  FORMAT                   = I4
  NOTE                     = "YEAR OF 1900"
END_OBJECT                 = YEAR

OBJECT                     = HOUR
  TYPE                     = VAX_INTEGER
  START_BYTE               = 3
  BYTES                    = 2
  FORMAT                   = I4
  NOTE                     = "HOUR OF YEAR STARTING AT 24"
END_OBJECT                 = HOUR

OBJECT                     = SECOND
  TYPE                     = VAX_INTEGER
  START_BYTE               = 5
```

BYTES	= 2
FORMAT	= I4
NOTE	= "SECOND OF HOUR"
END_OBJECT	= SECOND
OBJECT	= MILLI
TYPE	= VAX_INTEGER
START_BYTE	= 7
BYTES	= 2
FORMAT	= I3
NOTE	= "MILLISECOND OF SECOND"
END_OBJECT	= MILLI
OBJECT	= FDS_MOD16
TYPE	= VAX_INTEGER
START_BYTE	= 9
BYTES	= 2
FORMAT	= I5
NOTE	= "FDS MODULO 65536 COUNT"
END_OBJECT	= FDS_MOD16
OBJECT	= FDS_MOD60
TYPE	= VAX_INTEGER
START_BYTE	= 11
BYTES	= 2
FORMAT	= I2
NOTE	= "FDS MODULO 60 COUNT"
END_OBJECT	= FDS_MOD60
OBJECT	= FDS_LINES
TYPE	= VAX_INTEGER
START_BYTE	= 13
BYTES	= 2
FORMAT	= I3
NOTE	= "FDS LINE COUNT (1-800)"
END_OBJECT	= FDS_LINES
OBJECT	= INSTRUMENT_MODE
TYPE	= VAX_INTEGER
START_BYTE	= 15
BYTES	= 2
FORMAT	= I2
NOTE	= "DATA FORMAT MODE (0-31)"
END_OBJECT	= INSTRUMENT_MODE
OBJECT	= CHANNEL
ITEMS	= 16
TYPE	= VAX_INTEGER
START_BYTE	= 15



BYTES  
FORMAT  
NOTE  
END\_OBJECT  
END

= 2  
= I6  
= "UNCALIBRATED E-FIELD CHAN n"  
= CHANNEL

Pointer Label to an ASCII Table of Image Parameters:

```

NJPL1I0OPDS100000000    = SFDU_LABEL
/* This label describes the structure of the Index Table on each Voyager
/* Image CDROM. The table contains one row for each image file on the
/* CDROM.
RECORD_TYPE              = FIXED_LENGTH
FILE_RECORDS             = 6538
RECORD_BYTES             = 512
^IMAGE_INDEX_TABLE      = 'IMGINDEX.TAB'
OBJECT                   = IMAGE_INDEX_TABLE
FORMAT                   = ASCII
ROWS                     = 6538
ROW_BYTES                = 512
SFDU_LABEL               = NJPL1I0OPDS103347456
SPACECRAFT_NAME         = VOYAGER_2
MISSION_PHASE_NAME      = URANUS_ENCOUNTER
NOTE                     = "Flat Table File of Voyager Image Information"
OBJECT                   = SPACECRAFT_NAME
  TYPE                   = CHARACTER
  START_BYTE             = 1
  BYTES                  = 9
END_OBJECT

OBJECT                   = MISSION_PHASE_NAME
  TYPE                   = CHARACTER
  START_BYTE             = 13
  BYTES                  = 16
END_OBJECT

OBJECT                   = TARGET_NAME
  TYPE                   = CHARACTER
  START_BYTE             = 31
  BYTES                  = 8
END_OBJECT

OBJECT                   = IMAGE_ID
  TYPE                   = CHARACTER
  START_BYTE             = 39
  BYTES                  = 10
END_OBJECT

OBJECT                   = IMAGE_NUMBER      /* FDS COUNT
  TYPE                   = REAL
  START_BYTE             = 51
  BYTES                  = 8
END_OBJECT

```

OBJECT	= IMAGE_TIME
TYPE	= TIME
START_BYTE	= 61
BYTES	= 19
END_OBJECT	
OBJECT	= EARTH_RECEIVED_TIME
TYPE	= TIME
START_BYTE	= 81
BYTES	= 19
END_OBJECT	
OBJECT	= INSTRUMENT_NAME
TYPE	= CHARACTER
START_BYTE	= 101
BYTES	= 19
END_OBJECT	
OBJECT	= SCAN_MODE_ID
TYPE	= CHARACTER
START_BYTE	= 121
BYTES	= 7
END_OBJECT	
OBJECT	= SHUTTER_MODE_ID
TYPE	= CHARACTER
START_BYTE	= 129
BYTES	= 7
END_OBJECT	
OBJECT	= GAIN_MODE_ID
TYPE	= CHARACTER
START_BYTE	= 132
BYTES	= 7
END_OBJECT	
OBJECT	= EDIT_MODE_ID
TYPE	= CHARACTER
START_BYTE	= 145
BYTES	= 7
END_OBJECT	
OBJECT	= FILTER_NAME
TYPE	= CHARACTER
START_BYTE	= 153
BYTES	= 7
END_OBJECT	
OBJECT	= FILTER_NUMBER

```

TYPE                = INTEGER
START_BYTE         = 161
BYTES              = 7
END_OBJECT

OBJECT              = EXPOSURE_DURATION
TYPE                = REAL
START_BYTE         = 169
BYTES              = 7
END_OBJECT

OBJECT              = NOTE
TYPE                = CHARACTER
START_BYTE         = 177
BYTES              = 80
END_OBJECT

OBJECT              = IMAGE_VOLUME_ID
TYPE                = CHARACTER
START_BYTE         = 257
BYTES              = 8
END_OBJECT

OBJECT              = IMAGE_FILE_NAME
TYPE                = CHARACTER
START_BYTE         = 269
BYTES              = 48
END_OBJECT

OBJECT              = BROWSE_VOLUME_ID
TYPE                = CHARACTER
START_BYTE         = 317
BYTES              = 8
END_OBJECT

OBJECT              = BROWSE_FILE_NAME
TYPE                = CHARACTER
START_BYTE         = 329
BYTES              = 48
END_OBJECT

END_OBJECT          = IMAGE_INDEX_TABLE
END

```

## Binary Table of Voyager Magnetometer Values:

```

NJPL1IOOPDS111395180 = SFDU_LABEL
HARDWARE_TYPE         = VAX
RECORD_TYPE           = FIXED_LENGTH
RECORD_BYTES          = 24
FILE_RECORDS          = 474800
LABEL_RECORDS         = 50
OBJECT                = TABLE
  FORMAT              = BINARY
  ROWS                = 474750
  ROW_COLUMNS         = 5
  OBJECT              = TIME
  TYPE                = DOUBLE
  START_BYTE          = 1
  BYTES               = 8
  FORMAT              = 'F13.3'
END_OBJECT
OBJECT                = BX
  TYPE                = REAL
  START_BYTE          = 9
  BYTES               = 4
  FORMAT              = 'F12.4'
END_OBJECT
OBJECT                = BY
  TYPE                = REAL
  START_BYTE          = 13
  BYTES               = 4
  FORMAT              = 'F12.4'
END_OBJECT
OBJECT                = BZ
  TYPE                = REAL
  START_BYTE          = 17
  BYTES               = 4
  FORMAT              = 'F12.4'
END_OBJECT
OBJECT                = BT
  TYPE                = REAL
  START_BYTE          = 21
  BYTES               = 4
  FORMAT              = 'F12.4'
END_OBJECT
END_OBJECT
END

```

### L.3 SAMPLE IMAGE FILE LABELS

#### NOAA Elevation File:

```
NJPL1I0OPDS105529580      = SFDU_LABEL
RECORD_TYPE                = FIXED_LENGTH
RECORD_BYTES               = 1920
FILE_RECORDS               = 2880
LABEL_RECORDS              = 1
^ENGINEERING_HEADER        = 2
^IMAGE                      = 3
OBJECT                      = ENGINEERING_HEADER
  RECORDS                   = 1
END_OBJECT
OBJECT                      = IMAGE
  LINES                     = 2879
  LINE_SAMPLES              = 1920
  SAMPLE_BITS               = 8
END_OBJECT
FILE_NOTE                   = "
```

NOAA 30-second elevation averages were scaled to a range from 0 (sea level, lowest elevation) to 255 (highest elevation) so that each scaled value corresponds to a 15 meter interval. The data are arrayed in four files with the following longitudinal boundaries in degrees west longitude: 125 to 109, 109 to 100, 100 to 86, 86 to 66 degrees. Each file corresponds to a latitude range from 25 to 49 degrees north. Within each file each record corresponds to the northernmost latitude and the first sample in each record corresponds to the westernmost longitude. This file contains the Western region, 125 to 109 degrees west longitude."

END

Ocean Data System Image File:

```
NJPL1IOOPDS100263148      = SFDU_LABEL
/*  FILE CHARACTERISTICS
RECORD_TYPE                = FIXED_LENGTH
RECORD_BYTES               = 512
FILE_RECORDS               = 514
LABEL_RECORDS              = 2
OBJECT                     = IMAGE
  LINES                     = 512
  LINE_SAMPLES              = 512
  SAMPLE_BITS               = 8
END_OBJECT
/*  IMAGE DESCRIPTION
SPACECRAFT_NAME            = NIMBUS_7
TARGET_NAME                = EARTH
INSTRUMENT_NAME            = COASTAL_ZONE_COLOR_SCANNER
NOTE                       = "A CHLOROPHYLL CONCENTRATION IMAGE FROM THE
SANTA BARBARA AREA.  SCALE INCLUDED.  1 PIXEL IS ABOUT 1 KM
SQUARE. "
END
```

## Voyager Plasma Wave File:

This file is identified as an image in this example.

```
NJPL11OOPDS100822252      = SFDU_LABEL
RECORD_TYPE                = FIXED_LENGTH
RECORD_BYTES               = 1024
FILE_RECORDS               = 802
LABEL_RECORDS              = 2
^ENGINEERING_SUMMARY      = 3
^IMAGE                     = 4
OBJECT                     = ENGINEERING_SUMMARY
  BYTES                    = 1024
  STRUCTURE                = 'VGRPWS.LBL'
END_OBJECT
OBJECT                     = IMAGE
  LINES                    = 800
  LINE_SAMPLES             = 1600
  LINE_PREFIX_BYTES       = 220
  LINE_SUFFIX_BYTES       = 4
  SAMPLE_BITS              = 4
  SAMPLE_BIT_MASK         = 2#1111# /* NIBBLES IN TIME ORDER
END_OBJECT
SPACECRAFT_NAME            = VOYAGER_1
MISSION_PHASE_NAME        = JUPITER_ENCOUNTER
TARGET_NAME                = JUPITER_MAGNETOSPHERE
FRAME_ID                   = 16269.49
FRAME_PERIOD               = 48 <SECONDS>
SPACECRAFT_CLOCK_COUNT    = 16269.49 /* FLIGHT DATA SYSTEM (FDS)
SPACECRAFT_EVENT_TIME     = 1979/060-12:24:36 <UTC> /*FRAME BEGINNING
INSTRUMENT_NAME           = PLASMA_WAVE_SPECTROMETER
INSTRUMENT_MODE           = WAVEFORM_RECEIVER
INSTRUMENT_SAMPLING_RATE  = 28800 /* SAMPLES PER SECOND
INSTRUMENT_LOST_SAMPLES   = 128 /* LOST AT END OF EACH LINE
END
```



## Voyager CDROM Image File:

```

NJPL1I00PDS100000000      = SFDU_LABEL
/*      FILE FORMAT AND LENGTH
RECORD_TYPE                = VARIABLE_LENGTH
RECORD_BYTES               = 836
FILE_RECORDS               = 860
LABEL_RECORDS              = 54
/*      POINTERS TO STARTING RECORDS OF MAJOR OBJECTS IN FILE
^IMAGE_HISTOGRAM           = 55
^ENCODING_HISTOGRAM        = 57
^ENGINEERING_TABLE         = 60
^IMAGE                     = 61
SPACECRAFT_NAME            = VOYAGER_1
MISSION_PHASE_NAME         = SATURN_ENCOUNTER
TARGET_NAME                 = TITAN
IMAGE_ID                   = '1516S1-002'
IMAGE_NUMBER               = 34909.12 /*FLIGHT DATA SUBSYSTEM (FDS)
IMAGE_TIME                 = 1980-11-11T19:52:34Z
EARTH_RECEIVED_TIME        = 1980-11-11T21:19:46Z
NOTE                       = "ROUTINE MULTISPECTRAL LONGITUDE COVERAGE"
INSTRUMENT_NAME            = WIDE_ANGLE_CAMERA
SCAN_MODE                  = '3:1'
SHUTTER_MODE               = BOTSIM
GAIN_MODE                  = LOW
EDIT_MODE                  = '1:1' /*FULL RESOLUTION
FILTER_NAME                = CH4_JS
FILTER_NUMBER              = 0
EXPOSURE_DURATION          = 15.3600 <SECONDS>
/*      DESCRIPTION OF THE DATA OBJECTS CONTAINED IN FILE
OBJECT                     = IMAGE_HISTOGRAM
  ITEMS                    = 256
  ITEM_TYPE                = VAX_INTEGER
  ITEM_BITS                = 32
END_OBJECT
OBJECT                     = ENCODING_HISTOGRAM
  ITEMS                    = 511
  ITEM_TYPE                = VAX_INTEGER
  ITEM_BITS                = 32
END_OBJECT
OBJECT                     = ENGINEERING_TABLE
  BYTES                    = 242
  ^STRUCTURE               = 'ENGTAB.LBL'
END_OBJECT
OBJECT                     = IMAGE
  ENCODING_TYPE            = HUFFMAN_FIRST_DIFFERENCE
  LINES                    = 800
  LINE_SAMPLES             = 800

```

```
LINE_SUFFIX_BYTES      = 36
SAMPLE_TYPE           = UNSIGNED_INTEGER
SAMPLE_BITS           = 8
SAMPLE_BIT_MASK        = 2#11111111#
^LINE_SUFFIX_STRUCTURE = 'LINESUFX.LBL'
END_OBJECT
END
```

#### L.4 SAMPLE CUBE FILE LABEL

The following label is from a Sample Cube file. Note that just about everything necessary to interpret this data is included in the labels, including references to people with more information.

```
NJPL1I00PDS100000000    = SFDU_LABEL
/* File structure
RECORD_TYPE              = FIXED_LENGTH
RECORD_BYTES             = 512
FILE_RECORDS            = 5320
LABEL_RECORDS           = 8

/* Pointers to objects

^QUBE                    = 9

/* Qube object description

OBJECT                   = QUBE                /* object type serves as name

/* Data structure and description

STORAGE_TYPE            = BAND_SEQUENTIAL
AXES                    = 3
AXIS_ITEMS              = (128,128,83)
ITEM_BITS               = 16
ITEM_TYPE               = VAX_INTEGER        /* actually unsigned
AXIS_NAME               = (SAMPLE,LINE,BAND)
AXIS_UNIT               = (,,MICRONS)
AXIS_INTERVAL           = (,,0.)            /* discrete bands
AXIS3_BIN_CENTER_VALUE = (
    .7   .713 .725 .738 .751 .764 .776 .789 .802 .815
    .827 .84  .853 .866 .878 .891 .904 .917 .929 .942
    .955 .968 .98  1.006 1.031 1.057 1.082 1.108 1.133 1.158
    1.184 1.209 1.235 1.26  1.286 1.311 1.337 1.362 1.387 1.413
    1.438 1.464 1.489 1.515 1.54  1.566 1.591 1.617 1.642 1.668
    1.693 1.719 1.744 1.77  1.795 1.821 1.846 1.872 1.897 1.923
    1.948 1.974 1.999 2.025 2.051 2.076 2.102 2.127 2.153 2.178
    2.204 2.229 2.255 2.281 2.306 2.332 2.357 2.383 2.408 2.434
    2.46  2.485 2.511 )
CORE_ITEMS              = (128,128,83)
CORE_NAME               = "NORMALIZED REFLECTANCE"
CORE_UNIT               = DIMENSIONLESS
CORE_DIVISOR            = 10000
CORE_EXCLUDE_VALUE     = -32000

/* Instrument and target description

SPACECRAFT_NAME         = GALILEO
```

INSTRUMENT\_NAME = NIMS /\* Near Infrared Mapping Spectrometer  
TARGET\_BODY = GANYMEDE  
NOTE = "

GANCUB: Synthetic Ganymede Cube (17mar87)

(by Robert Singer, with additions by Roger Clark and Bob Mehlman)  
GANCUB is a synthetically generated 3-dimensional data set (image cube) of the type to be returned by mapping spectrometers such as the NIMS instrument on Galileo. The spatial information (the first two array dimensions) is based on a small section of a Voyager image of Ganymede, taken in the longest wavelength filter (orange, I believe).

The third (spectral) dimension consists of one spectrum per spatial pixel. These spectra originated from 25-30 near-IR laboratory spectra (~0.7 to 2.5 um) of ice, minerals, and ice-mineral mixtures, and were converted to NIMS wavelengths. A small amount of random noise was added to the lab spectra before assignment to each pixel in the image to more closely simulate actual NIMS data, but the net effect is still quite smooth, probably smoother than will be realized with actual spacecraft measurements. A spectrum was assigned for each pixel based on the closest match between Voyager orange albedo and the ~0.7 um reflectance of the spectra. (These albedos did not vary much so they were stretched to fill the range.) For the ice spectra, this depends most heavily on the grain size and level of contamination. Some clay minerals and amorphous hydrous weathering products (palagonites) are also spectrally represented. There are small exposures of olivine and pyroxene as well, for variety.

Most data words in the file range from 0 to 10,000, corresponding to normalized spectral values from 0. to 1.0 with a significance of about .002. There are a few small negative values. There are also 1437 fill values of -32,000 corresponding to missing pixels in the lab spectra.

References:

Roger N. Clark  
U.S. Geological Survey, Mail Stop 964  
Box 25046, Federal Center  
Denver, CO 80225  
Phone: 303/236-1332, x1212 (secy), x1411  
Telemail: RNCLARK

Robert Singer  
Lunar and Planetary Laboratory  
Space Sciences Bldg.  
University of Arizona  
Tucson, AZ 85721  
Phone: 602/621-4573

Bob Mehlman (software)  
UCLA/IGPP  
Los Angeles, CA 90024  
Phone: 213/825-2434, -3123  
Telemail: RMEHLMAN  
SPAN: GRUMPY::RMEHLMAN, ISSAC::RMEHLMAN "

END\_OBJECT  
END



## Appendix M

### PDS CODING STANDARDS

The purpose of these standards is to facilitate the creation of highly readable and maintainable code. This standard will:

- (1.) Increase portability to a new host environment.
- (2.) Allow mechanical processing of computer programs (i.e., a documentation reader).
- (3.) Provide for new personnel to learn easily about existing software.
- (4.) Facilitate reuse of program segments.

It is important for the software engineer to understand the philosophy behind the standards. The following coding standards are not absolute rules, nor are they meant to hinder the engineer's productivity. By following standards, the goals listed above can be obtained, leading to a more maintainable and readable product. Moreover, these Standards are not exhaustive; additions to the Standard shall occur throughout the development phase. Therefore, it is just as important for the software engineer to follow the spirit of the Standard as it is to follow the letter of the Standard.

The following code development standards address comments, programming style, explicit typing, naming conventions, language specific practices, common software, and software engineering notebooks. Each guideline is followed by a brief paragraph motivating the guideline. Occasionally, a stylistic motivation is given which indicates that the choice was arbitrary; however, the choice shall be used to keep the code consistent.

PDS application software shall be organized as a hierarchy of software components:

- (1.) Function
- (2.) Program
- (3.) Segment
- (4.) Unit
- (5.) Module

In this hierarchy, a function is the largest and most general component and a module is the smallest and most detailed.

Each software component shall have the following format:

Component (parameter1, parameter2, ...) followed by:

- (1.) Header block
- (2.) Parameter declarations and descriptions
- (3.) Local variable declarations and descriptions
- (4.) Code body

## M.1 COMMENTS

Comments shall occur at the following places in a software component:

- (1.) A header which is surrounded by the comment symbol as a block shall highlight the beginning of a software component. Additionally, it is recommended that the higher level software component should contain one more boundary line on both the top and bottom sides than the next lower level. This header format applies to all levels of software components.

```
*****
*
* Component subname [(var1[,var2[...]])]
* Change history : (date, author, summary)
* Overview : (what it does)
* Invocation example :
* Detailed description : (processing logic)
* Internal/external references :
* Author and institution
* Version and date :
* Logical--units on input
* Logical--units on output
* Limitations
*****
```

Motivation: The header contains the essential sections necessary to document a software component. These sections are formatted so that the documentation reader can extract them. Moreover, the format chosen separates the header from the rest of the software component.

- (2.) Logical blocks of code shall be commented in block format. The majority of components should be decomposable into blocks. A typical block may be a nested if block, loop, or procedure initialization statements. Comments describing logical blocks of code shall be surrounded by starred boxes for emphasis.

Motivation: Code is described best in logically cohesive blocks than by many over-commented lines. Boxing in comments aids visually in the separation of blocks throughout the code.

- (3.) Comment blocks at the same software component level shall be aligned on the same column. In general, comments of individual statements shall be aligned, especially for those statements logically related. Individual lines of code shall be commented only when the comment does more than just echo the statement coded. In general, it should not be necessary to comment many individual lines of code, since most statements belong to a logical block. The following shall be a guide to comment individual lines of code:

- (a.) The statement performs a complete function.
- (b.) The programmer determines that specific information is required for general software understandability.
- (c.) The comment is required to explain programming language or structure construct idiosyncrasies.



- (d.) The programmer intends to further clarify nesting levels by commenting the beginning and ending statements of a particular block of code.
- (e.) The statement is setting default values or error flags where additional comments are required to state the function.

Motivation: Over-commenting ruins the readability of code and burdens the reader with unnecessary details.

- (4.) Comments shall reflect what the code is doing.

Motivation: Inaccurate comments are worse than no comments. An inaccurate description of the code can become a major problem in debugging and maintaining code, especially when the reader relies on descriptions instead of actual code.

- (5.) The structure of the code shall be directly traceable from the structure of the design.

Motivation: The traceability of the code from structure charts is a direct check on the design process. If the code deviates drastically from the charts, this signals that either the coding was done incorrectly, or that the design is faulty.

- (6.) Local variables shall be described and declared in a separate section before any executable statements by a line or two of description.

Motivation: Similar to describing parameters, local variables must also be described and declared together to complete the documentation of the software component.

## M.2 PROGRAMMING STYLE

Stylistic issues address choosing a single consistent way of performing an operation when there are multiple acceptable solutions. Stylistic conventions may seem to burden the programmer especially when his style does not agree with the imposed Standard. However, stylistic conventions allow easier mechanical processing of programs if the conventions are followed. Moreover, a standard programming style enhances program readability, since the reader doesn't need to readjust to a different programming style.

- (1.) The recommended size of a PDS module shall be 100 lines of code or less. Programmers shall follow the following rule that a component shall have one purpose, described in the overview section of the header. Moreover, the overview shall not consist of a compound subject (i.e., no "and" in it). If it does, then the component shall be split.

Motivation: Limiting the scope of a subroutine to one purpose maximizes program modularity and cohesion.

- (2.) Multiple entry and exit points shall not be allowed.

Motivation: Multiple entry and exit points go against the philosophy of structured programming. If the structured design is done correctly, the code should reflect one entry and one exit point. In general, multiple exit points are unnecessary, since flag setting and branching can often solve the problem and multiple entry points can be replaced by writing separate routines.

- (3.) The use of *goto* statements shall be limited.

Motivation: Extensive use of *goto* statements goes against the philosophy of structured design and programming. Its use contributes to a non-top-down programming style that can be best described as spaghetti code. A common instance where the legitimate use of *gotos* may occur is: branching to an error or exit routine, so that multiple return statements are avoided.

- (4.) Indentation shall reflect the logical structure of the code. Levels of nesting shall be set apart by four columns per indentation. Moreover, each statement that ends a structure shall appear in the same column as the statement that started the structure. Example:

```
do var = e1, e2, e3
  statement 1
  statement 2
  statement 3
end do
```

Motivation: Code that is indented well reflects the logic of a program much better than non-indented code. Four columns is recommended for the length of indentation, since it is recognized easily and does not waste line space.

- (5.) Constants shall be defined through the use of C *define* statements or FORTRAN *parameter* statements, and contained in separate include files. The use of numerals in the code shall be kept to a minimum. Exception: When initializing variables, it is easier to use numerals.

Motivation: By defining constants and maintaining them in include files, individual programs will not need to be modified due to changed constants.

- (6.) Blank lines shall be used to separate code from comments.

Motivation: This improves readability.

- (7.) Spaces shall be used to separate variables in parameter lists, equations, and other programming statements.

Motivation: This improves readability.

- (8.) If a statement is too long to fit on a line comfortably, the continued statement shall be indented the same amount as the previous line.

Motivation: This improves readability.

- (9.) When code is commented out, additional comments shall be added telling when and why the code has been changed. Commented out code shall be set off by marks, i.e., **\*\* old \*\***.

Motivation: Many times code is commented out on the fly and therefore not well documented. Code should be documented at all stages of development including its maintenance phase. Commented out code shall be made distinguishable from code that is in use.

- (10.) Complicated expressions shall be transformed into several simple expressions aligned on separate lines.

Motivation: The C language allows the programmer to encode complicated and efficient expressions at the expense of clarity and readability. To help future readers understand the entire expression it is appropriate to break the expression down into small, clear subexpressions. This can also provide more space for individual comment.

### M.3 EXPLICIT TYPING

All variables and functions shall be explicitly typed. Each variable shall be declared separately by the use of a type statement, i.e., real, integer, etc.

Motivation: Many languages consider explicit typing a standard feature. This feature virtually eliminates bugs due to the failure of declaring variables.

## M.4 NAMING CONVENTIONS

- (1.) Variable name and file name length shall be dictated by the machine.

Motivation: PDS does not put a restriction of the size of a variable or file name. Therefore, the maximum variable name size shall be determined by the limitation of the machine.

- (2.) All global variable and software component names shall be unique in the first six characters.

Motivation: This ensures unique names for global variables and software components.

- (3.) A software component naming scheme shall exist which categorizes components into appropriate programming areas.

Example: Prefixes in software component names: `slibgetline` or `slib_get_line`, where the prefix `slib` tells this routine it belongs to the system library.

Motivation: The scheme shall facilitate the identification of software components throughout the system.

- (4.) The following file extensions shall be used:

- (a.) `.FOR` – FORTRAN source code files
- (b.) `.INC` – FORTRAN include file
- (c.) `.C` – C source code files
- (d.) `.H` – C include file
- (e.) `.COM` – VAX DCL command files
- (f.) `.OBJ` – VAX object files
- (g.) `.EXE` – VAX executable files
- (h.) `.MAR` – VAX MACRO assembly code files

Motivation: The use of standard terminology facilitates the system build process.

- (5.) Descriptive variable names shall be used. One or two character variable names shall be avoided.

Exception: array indices, loop counter variables, etc.

Motivation: Comprehension of variables is enhanced.

## M.5 LANGUAGE SPECIFIC PRACTICES

The following sections define the language specific coding standards for the FORTRAN and C programming languages.

### M.5.1 PDS FORTRAN Coding Standards

- (1.) In general, PDS applications software written in FORTRAN shall be restricted to the ANSI FORTRAN 77 (X3.9-1978) Standard to make the code more transportable and consistent. Extensions to the ANSI FORTRAN 77 Standard involving do loops, do while loops, include files, and lower case shall be available to programmers. These extensions show the structure of the code more clearly than the ANSI counterparts. Since applications shall be shared between Nodes, transportability shall be a major concern. Therefore, a precompiler shall be provided in order to translate all non-ANSI extensions into ANSI FORTRAN 77.

PDS EXTENSIONS to ANSI FORTRAN 77:

**Common and Parameter Statements.** *Common* and *Parameter* statements shall be placed in separate include files. These files shall be included by means of the *include* statement in any software component that uses the *Common* or *Parameter* statements. The syntax of the *include* statement shall be *include filename/list*. The */list* qualifier is optional and directs the compiler to list the include file with the rest of the source file in the list file. These two types of include files shall be distinguishable by using the prefixes *com\_* and *par\_* followed by the filename.inc. The *com\_filename.inc* shall contain the type declarations of all the *common* block variables followed by the declaration of the *common* block itself. The *par\_filename.inc* shall contain alternating type declarations and *Parameter* statements.

**Motivation:** Ensures that the declarations of a *common* block will be unique, since all references made to it shall be through the declaration in the include file.

**Implicit None.** The *implicit none* statement shall be used in the declaration section.

**Motivation:** This helps find any variables that are not explicitly declared.

**Unnamed Common.** Unnamed *Common* shall not be used.

**Motivation:** The use of unnamed *Common* is incompatible in a large user group. Using two or more software components containing unnamed *Common* blocks with different contents can cause unpredictable results.

**Labeled Loops.** Do loops without statement labels shall be preferred over labeled do loops.

**Example of:** do loops without labels:

do var = e1, e2, e3	do for var = e1, e2, e3
.	.
.	.
.	.
end do	end for

**Example of:** do loops with labels:

```
do label var = e1, e2, e3
.
.
.
label end do
```

**Motivation:** It delineates the structure of the program and avoids the use of statement labels.

**Do While.** Do while loops shall be preferred over labeled *if* blocks.

**Example of:** do while loops:

do while var = e1, e2, e3	do while var = e1, e2, e3
.	.
.	.
.	.
end do	end while

**Motivation:** While loops express the structure of a program better than labeled *if* blocks.

**Block If.** The *block if* construct shall be preferred over the logical *if () then*:

block if:	logical if:
-----------	-------------

```

if () then
    if statements
else
    else statements
end if

```

```

if () then
    if statements
if not() then
    else statements

```

Motivation: The *block if* statement is more concise and easier to follow.

FORTRAN shall be written in either upper- or lowercase.

Motivation: Only the uppercase character set is defined in the ANSI FORTRAN 77 standard. Lowercase FORTRAN statement will be translated into uppercase.

## M.5.2 PDS C Coding Standards

This section provides standard coding guidelines for programs written in C. In order to make the C applications code as portable as possible, VMS-unique system features shall not be used in PDS applications code.

In addition to the following guidelines, the standard reference for coding in C shall be *THE C PROGRAMMING LANGUAGE* by Kernighan and Ritchie, (Prentice-Hall,1978.)

- (1.) Structure member names. All structure member names shall be unique.

Motivation: The standard C definition does not bind a member name to the structure that contains it.

- (2.) Structure/Union assignment. Not all compilers allow structure/union assignment. To accommodate differences in compilers, the following shall be used: Define a symbol (STRASS) "structure assignment" in system.h to indicate whether structure/union assignment is allowed. Example:

```

struct str a,b;

#if STRASS
    a = b;
#else
    bytncpy( (char*)a, (char *)b, sizeof(str) );
#endif

```

(bytncpy is a function that copies a given number of bytes from b to a.)

- (3.) Passing a structure or a union as a function argument. Structures or unions shall not be passed as functional arguments. The & operator shall be used to pass a pointer to them instead.

Motivation: This shall enhance portability; structure and unions may be passed as an argument only in newer compilers.

- (4.) Labels and gotos. In cases where a goto is unavoidable, labels shall be in upper case, and placed starting in column 1 with the following colon on the same line.

Example: goto OUTLOOP;

OUTLOOP: statement;

Motivation: Stylistic.

- (5.) Typedef. New data type names created by using the *typedef* statement shall be in upper case.

Example: *typedef int* LENGTH;

Motivation: Stylistic.

- (6.) Pointer array. Pointer array arguments shall be defined as \*x[ ], not \*\*x. Pointer array variables shall be defined as \*\*x.

Motivation: Stylistic

- (7.) Character set differences. Dependence on the ASCII character code set shall be avoided; use standard library functions to determine character type, and perform upper/lower case conversion. One may assume that the characters '0' through '9', 'a' through 'z', and 'A' through 'Z' are consecutive.

Motivation: This feature enhances portability.

- (8.) Allocation of storage for character strings. Character string allocation shall use the form "MAXLEN+1" to emphasize the real length of the string including the end of string terminator at the end.

Example: Allocate storage for a character string of 80 characters:

```
# define MAXLEN 80
char str[MAXLEN+1]
```

Motivation: Stylistic.

- (9.) Octal vs. hexadecimal constants. If binary representation is needed, hexadecimal constants shall be used instead of octal.

Motivation: Hexadecimal constants are more suited for modern 16- and 32-bit computer architectures.

- (10.) *Unsigned char* and *unsigned long*. *Unsigned char* and *unsigned long* shall not be used.

Motivation: Portability. *Unsigned char* and *unsigned long* are not available on some compilers.

- (a.) Pointer arguments. All pointer arguments to a function shall be cast to the proper type.

Example: the standard function *strcpy* takes two *char* pointers:

```
strcpy( (char *)dest, (char *)source );
```

Motivation: The internal forms (or even the size) of pointers to different objects may be different. Therefore, it is safer to use the cast operator.

- (11.) Pointer arithmetic. Pointer arithmetic shall be limited to a single dimensioned arrays.

Motivation: Portability. In general, the numeric value of a pointer is CPU dependant.

- (12.) Pointer conversion. Pointer conversion shall be explicitly stated by a cast operator.

Motivation: Avoids future programming problems. This guideline agrees in spirit of explicitly declaring variables.

- (13.) Type sizes. Be aware that the size of pointer and *int* types are not always the same. (Example: VAX/VMS C uses 2 bytes for *int* and 4 bytes for pointers.) Care must be taken to use the correct size. Ensure that arguments and function return values are of the correct size.

Motivation: Portability. Programs developed on systems in which *int* and pointer types are the same size fail to work when ported to systems with different sizes.

- (14.) Placement of braces. The two braces { } used to delimit a compound statement shall be indented on the same column. Example:

```
    if (i > 0 )
    {
        if-part;
    }
    else
    {
        else-part;
    }
```

Motivation: It is easier to trace a program with matching levels of delimiters (braces) than one with uneven levels.

- (15.) Function definitions. If a function returns anything, then what is returned shall be explicitly defined, even if it is the default type, *int*.

Motivation: This features increases program readability and forces the programmer to think about the type of value the function shall return.

- (16.) Assignment operators. The following obsolete assignment operators shall not be used: =+, =--, =\*, =/, =%, =>>, =<<, =&, =;, =|

Motivation: These assignment operators are obsolete and have been replaced by *operator* = type operators.

- (17.) Conditionally compiled integration code. All temporary statements which will only survive to integration testing shall be surrounded by the following conditional compilation control lines:

```
    #ifdef integration
        statements;
    #endif
```

By inserting or not inserting the statement *#define integration* at the beginning of the source file or in an include file, the integration statements can be compiled or not compiled.

Motivation: This provides programmers with a standard method of introducing temporary code into the integration environment without affecting the standard code.

- (18.) Abbreviated variable names shall be separated by underscore. Example: *max\_int* for maximum integer.

Motivation: Readability of variables is enhanced.

## M.6 VALID CHARACTER SET

This section contains the valid character set that must be used in the construction of PDS terms.

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1 2 3 4 5 6 7 8 9 0 \_

