INTERNATIONAL STANDARD

ISO 14961

Second edition
2002-09-15

Space data and information transfer systems — Parameter value language specification

Systèmes de transfert des informations et données spatiales — Spécification d'un langage à valeurs paramétriques

Reference number
ISO 14961:2002(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 14961 was prepared by the Consultative Committee for Space Data Systems (CCSDS) (as CCSDS 641.0-B.2, June 2000) and was adopted (without modifications except those stated in clause 2 of this International Standard) by Technical Committee ISO/TC 20, *Aircraft and space vehicles*, Subcommittee SC 13, *Space data and information transfer systems*.

This second edition cancels and replaces the first edition (ISO 14961:1997), which has been technically revised.

Space data and information transfer systems — Parameter value language specification

1 Scope

This International Standard specifies the requirements for a parameter value language specification for space data and information transfer systems.

The scope and field of application are furthermore detailed in subclauses 1.1 and 1.2 of the enclosed CCSDS publication.

2 Requirements

Requirements are the technical recommendations made in the following publication (reproduced on the following pages), which is adopted as an International Standard:

CCSDS 641.0-B.2, June 2000, *Recommendation for space data system standards — Parameter value language specification (CCSD0006 and CCSD0008)*.

For the purposes of international standardization, the modifications outlined below shall apply to the specific clauses and paragraphs of publication CCSDS 641.0-B.2.

Pages i to v

This part is information which is relevant to the CCSDS publication only.

Page 1-4

Replace the reference indicated by the following:

[2] ISO/IEC 8824-1:1998, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation — Part 1* and its amendments ISO/IEC 8824-1:1998/Amd 1:2000 and ISO/IEC 8824-1:1998/Amd 2:2000.

Add the following information to the reference indicated:

[3] Document CCSDS 301.0-B-2, April 1990, is equivalent to ISO 11104:1991.

Page C-1

Replace the reference indicated by the following:

[2] ISO 8859-1:1998, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*.

3 Revision of publication CCSDS 641.0-B.2

It has been agreed with the Consultative Committee for Space Data Systems that Subcommittee ISO/TC 20/SC 13 will be consulted in the event of any revision or amendment of publication CCSDS 641.0-B.2. To this end, NASA will act as a liaison body between CCSDS and ISO.

Consultative Committee for Space Data Systems

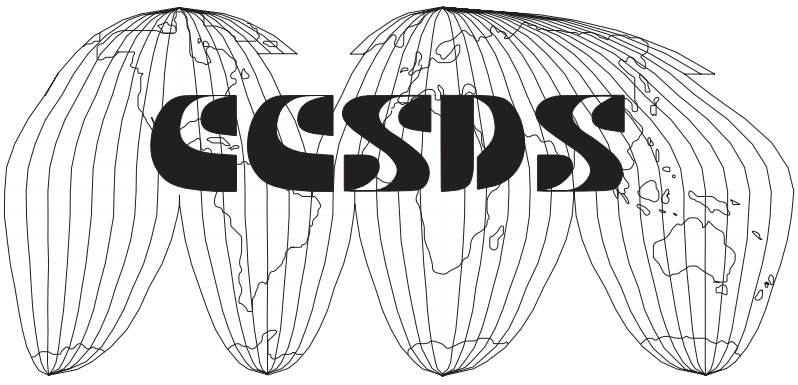
RECOMMENDATION FOR SPACE DATA SYSTEM STANDARDS

PARAMETER VALUE LANGUAGE SPECIFICATION (CCSD0006 and CCSD0008)

CCSDS 641.0-B-2

BLUE BOOK

June 2000



(Blank page)

AUTHORITY

Issue:	Blue Book, Issue 2
Date:	June 2000
Location:	Toulouse, France

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS Recommendations is detailed in Reference [C3], and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This Recommendation is published and maintained by:

CCSDS Secretariat
Program Integration Division (Code MT)
National Aeronautics and Space Administration
Washington, DC 20546, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of member space Agencies. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommendations** and are not considered binding on any Agency.

This **Recommendation** is issued by, and represents the consensus of, the CCSDS Plenary body. Agency endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever an Agency establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommendation**. Establishing such a **standard** does not preclude other provisions which an Agency may develop.

- Whenever an Agency establishes a CCSDS-related **standard**, the Agency will provide other CCSDS member Agencies with the following information:

  -- The **standard** itself.

  -- The anticipated date of initial operational capability.

  -- The anticipated duration of operational service.

- Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommendation** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommendation** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or, (3) be retired or canceled.

In those instances when a new version of a **Recommendation** is issued, existing CCSDS-related Agency standards and implementations are not negated or deemed to be non–CCSDS compatible. It is the responsibility of each Agency to determine when such standards or implementations are to be modified. Each Agency is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommendation.

FOREWORD

This document is a technical Recommendation for the specification of the Parameter Value Language (PVL) and has been prepared by the Consultative Committee for Space Data Systems (CCSDS).

This Recommendation defines the Parameter Value Language that provides a human readable, machine processable language for naming and expressing data values. It allows implementing organizations within each Agency to proceed coherently with the development of compatibly derived Standards for space data systems and widely dispersed data users that are within their cognizance. Derived Agency Standards may implement only a subset of the optional features allowed by the Recommendation and may incorporate features not addressed by the Recommendation.

Through the process of normal evolution, it is expected that expansion, deletion, or modification to this document may occur. This Recommendation is therefore subject to CCSDS document management and change control procedures as defined in Reference [C3]. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relative to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- National Aeronautics and Space Administration (NASA)/USA.
- National Space Development Agency of Japan (NASDA)/Japan.
- Russian Space Agency (RSA)/Russian Federation.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Communications Research Laboratory (CRL)/Japan.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Federal Service of Scientific, Technical & Cultural Affairs (FSST&CA)/Belgium.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Industry Canada/Communications Research Centre (CRC)/Canada.
- Institute of Space and Astronautical Science (ISAS)/Japan.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- National Space Program Office (NSPO)/Taipei.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title and Issue	Date	Status
CCSDS 641.0-B-1	Parameter Value Language Specification (CCSD0006), Issue 1	May 1992	Original Issue (superseded).
CCSDS 641.0-B-2	Parameter Value Language Specification (CCSD0006 and CCSD0008), Issue 2	May 1999	Current Issue. - Format updated for consistency with current CCSDS style guidelines. - Specification of PVL character set updated to include G1 set of ISO 8859-1 to support use of accented characters used in many languages. - Purpose and Applicability updated to include usage outside of CCSDS. - ASN.1 corrected to ensure order of PVL statements is maintained. - ASN.1 DayOfMonth corrected. - Conformance Section added. - Constraints Section added. - Added clarification that alternate syntactic forms should not be used to carry meaning. - Updated Numeric and White Space definitions. - Consistently capitalized words used to denote PVL syntactical elements.

NOTE – Substantive changes from the previous issue are indicated with change bars in the outside margin.

CONTENTS

CONTENTS (continued)

Figure Page

Table

(Blank page)

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

The purpose of this document is to establish a common Recommendation for the specification of a standard keyword value type language for naming and expressing data values. It is useful for wider audiences, but it is designed be used to interchange data in a more uniform fashion within and among Agencies participating in the Consultative Committee for Space Data Systems (CCSDS). This Recommendation provides an overview and formal syntax specification of the Parameter Value Language (PVL). Two versions of PVL are defined—the basic version (CCSD0006) and an extended character set version (CCSD0008).

1.2 APPLICABILITY

The specifications in this document are applicable to applications where a keyword value language is desired. The specifications in this document shall be invoked through the normal standards program of Agencies participating in CCSDS and are applicable to all space-related science and engineering data exchanges where a keyword value language is desired.

1.3 RECOMMENDED APPROACH TO READING THE DOCUMENT

A proper understanding of this Recommendation requires familiarity with the terminology used in this document. Terms are defined as they are introduced in the text. Individuals who are accessing the document out of sequence may wish to refer to 1.4, which presents the terminology used in this document, and Annex A, which presents a summary of the acronyms used in this document. Reference [C5] is a tutorial which describes the requirements, the techniques used to fulfill the requirements, usage guidelines and parser implementation guidelines for PVL. Some readers may find it useful to read Reference [C5] prior to reading this document.

The document is structured as follows:

- Section 2 describes the PVL language, using English text and syntax diagrams.
- Section 3 describes constraints that must be followed to successfully exchange PVL objects.
- Section 4 provides the formal syntax specification written in Abstract Syntax Notation One (ASN.1, see Reference [2]). The comments in the ASN.1 are part of the specification. This is the ruling form of the specification.
- Section 5 describes the single conformance level for this specification.
- Annex A contains acronyms used in this document.
- Annex B lists the ASCII codes for the characters used in PVL.
- Annex C contains a list of informative references.

This document uses syntax diagrams to illustrate the syntax of the various language constructs. Components of the construct are called elements, are presented in boxes or circles and are connected by directional lines. The following conventions are used:

– Elements that are presented in uppercase and lowercase letters in rectangles are defined elsewhere in the document.

– Elements that are presented in a circle as a single bold character are delimiters or reserved characters.

– Elements that are presented in lowercase letters in a rectangle with rounded corners are basic items not further defined in the syntax diagrams of this document.

– Elements that are presented in bold characters in a rectangle with rounded corners are keywords.

– The item named on the left of the ::= symbol is the item being defined.

– The diagram on the right of the ::= symbol is the definition.

– A vertical branch represents a choice.

– A repetition is indicated by a loop back covering the object to be repeated.

– The termination of each structure is represented by the o symbol.

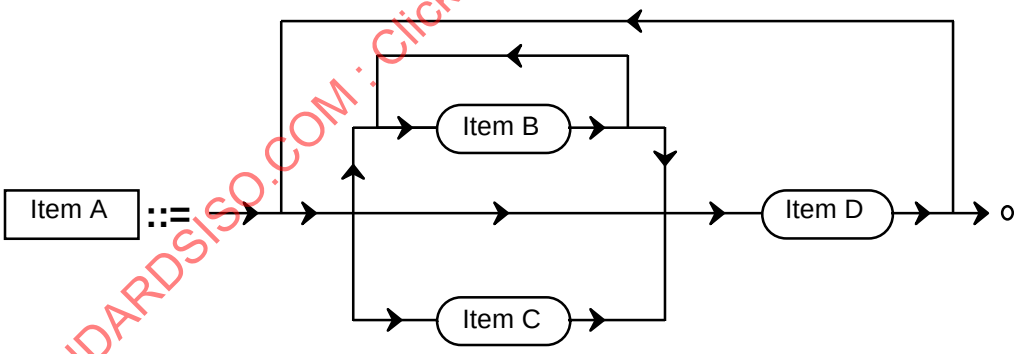


Figure 1-1: Example Structure Diagram

For example:

In this example Item A is defined as first a choice between Items B or C or nothing, where Item B itself may be repeated any number of times. Then this structure is followed by one Item D. Once this structure is built up, it may then all be repeated any number of times, until the choice to pass onto the o symbol is taken. Of course if any items on the right (B, C or D) are an Item A or contain an Item A, the definition is recursive. Readers are warned to watch for recursive structure definitions, which are permitted in this Recommendation.

1.4 DEFINITIONS

1.4.1 TERMINOLOGY

Aggregation Block: A named collection of Assignment Statements and/or other Aggregation Blocks.

Alphanumeric character set: The set of characters comprised of the digits 0 through 9 and the letters a-z or A-Z.

Block Name: The name used to identify an Aggregation Block.

Comment: A delimited string of characters, which is treated as White Space syntactically. Comments are intended to provide explanatory information.

Comment delimiters: The character pairs (/* and */) used to delimit a Comment.

End Statement: An optional statement that terminates the PVL Module prior to the end of the provided octet space.

Numeric: A sequence of characters that conform to encoding rules that permit its interpretation as a number.

Octet: A sequence of eight bits.

Parameter Name: The name used to reference the value assigned in the Assignment Statement.

PVL Module: The externally defined octet space that may be optionally terminated by a PVL End Statement, within which PVL statements are written.

Quote String Delimiters: The symbols apostrophe or quotation mark.

Quoted String: Zero or more PVL Characters enclosed between matching Quote String Delimiters.

Reserved Characters: The set of PVL Characters that may not occur in Parameter Names, Unquoted Strings, or Block Names.

Sequence: A delimited collection of values in which the order of the enclosed values is significant.

Set: A delimited collection of values in which the order of the enclosed values is not significant.

Standard Formatted Data Unit: Data units that conform to a specific set of CCSDS Recommendations.

Unquoted String: A value consisting of a sequence of Unrestricted Characters.

Unrestricted Characters: The set of PVL Characters that may be used to form Parameter Names, Unquoted Strings, or Block Names.

White Space: One or more space or format effector characters. Used to separate syntactic elements and to promote readability between syntactic elements or within the contents of Comment or text strings.

1.5 NORMATIVE REFERENCES

[1] *Information Processing — Representation of Numerical Values in Character Strings for Information Interchange*. International Standard, ISO 6093-1985(E). Geneva: ISO, 1985.

[2] *Information Technology — Open System Interconnection — Specification of Abstract Syntax Notation One (ASN.1)*. International Standard, ISO/IEC 8824:1990(E). 2nd Ed. Geneva: ISO, 1990.

[3] *Time Code Formats*. Recommendation for Space Data Systems Standards, CCSDS 301.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, April 1990 or later issue.

2 OVERVIEW OF THE LANGUAGE

2.1 CHARACTER SET DEFINITIONS

The following sections contain character set definitions used in this specification. A clear understanding of these terms is necessary to understand this Recommendation.

2.1.1 PVL CHARACTER SET

The PVL Character Set is split into three subsets: White Space Characters, Reserved Characters, and Unrestricted Characters. The CCSD0006 version of the PVL Character Set is a subset of the ASCII character set. The specific subset is shown in Annex B. The CCSD0008 version of the PVL Character Set is the CCSD0006 version of the PVL Character Set with the Additional Character Set.

2.1.1.1 White Space Character Set

The White Space Character Set is defined as the following characters: space, carriage return, line feed, horizontal tab, vertical tab, and form feed. A sequence of one or more or these characters is known as White Space. The semantic effect of White Space between syntactic elements is not affected by its length.

NOTE – Since sequences of one or more of any of the White Space Characters between syntactic elements are syntactically equivalent, the number of White Space Characters or the use of a particular White Space Character may not be used to provide different meanings (semantics) for applications.

2.1.1.2 Reserved Character Set

The Reserved Character Set is a collection of characters reserved for specific purposes or future use. The Reserved Character Set is defined in Table 2-1.

Table 2-1: Reserved Character Set

Symbol	Name	Symbol	Name	Symbol	Name
&	Ampersand	[	Left Square Bracket	%	Percent Sign
<	Less-Than Sign (Open Angle Bracket)	]	Right Square Bracket	+	Plus Sign
>	Greater-Than Sign (Close Angle Bracket)	=	Equal Sign	"	Quotation Mark
'	Apostrophe	!	Exclamation Point	;	Semicolon
{	Left Curly Bracket (Left Brace)	#	Number Sign, (Hash)	~	Tilde
}	Right Curly Bracket, (Right Brace)	(	Left Parenthesis	|	Vertical Line
,	Comma	)	Right Parenthesis		

2.1.1.3 Unrestricted Character Set

The Unrestricted Character Set is a collection of PVL Characters that are not reserved or used as White Space. The CCSDS0006 version of the Unrestricted Character Set is defined as the alphanumeric character set (`a-z`, `A-Z`, and `0-9`) and the non-alphanumeric characters in Table 2-2. The CCSD0008 version of the Reserved Character set is the CCSD0006 version of the Unrestricted Character Set extended by the Additional Character Set.

Table 2-2: CCSD0006 Unrestricted Character Set

Symbol	Name	Symbol	Name	Symbol	Name
a-z	Lower Case Alphabetics	A-Z	Upper Case Alphabetics	0-9	Digits
*	Asterisk	$	Dollar Sign	?	Question Mark
^	Circumflex Accent, (Caret)	`	Grave Accent	/	Solidus, (Forward Slash)
:	Colon	.	Full Stop, (Period)	\	Reverse Solidus, (Backward Slash)
@	Commercial At	-	Hyphen-Minus Sign	_	Low Line, (Underscore)

2.1.1.4 Additional Character Set

The Additional Character Set is defined as the G1 character set of ISO 8859-1. The Additional Character Set is shown in Table 2-3.

Table 2-3: Additional Character Set

NBSP	°	À	Ð	à	ð
¡	±	Á	Ñ	á	ñ
¢	²	Â	Ò	â	ò
£	³	Ã	Ó	ã	ó
¤	´	Ä	Ô	ä	ô
¥	µ	Å	Õ	å	õ
|	¶	Æ	Ö	æ	ö
§	·	Ç	×	ç	÷
¨	¸	È	Ø	è	ø
©	¹	É	Ù	é	ù
ª	º	Ê	Ú	ê	ú
«	»	Ë	Û	ë	û
¬	¼	Ì	Ü	ì	ü
SHY	½	Í	Ý	í	ý
®	¾	Î	Þ	î	þ
¯	¿	Ï	ß	ï	ÿ

2.1.2 COMMENT

A Comment consists of zero or more PVL Characters enclosed between a pair of Comment Delimiters. The Begin Comment Delimiter is the forward slash-asterisk sequence (`/*`). The End Comment Delimiter is the first following asterisk-forward slash sequence (`*/`). Comments are treated the same as White Space when occurring between syntactic elements, except that Comments cannot appear within a Units Expression. Comments shall not be embedded within other Comments.

NOTE – Since Comments are normally syntactically equivalent to White Space, the presence or absence of Comments may not be used to provide different meanings (semantics) for applications.

2.2 LANGUAGE SYNTAX

PVL provides a specific syntax for the association of values with parameters. A PVL Module consists of a sequence of zero or more statements. These statements are found within an externally provided sequence of Octets. Some or all of these statements can be aggregated into named blocks. Layout (i.e., the use of White Space to promote human readability) is not significant for the interpretation of these statements.

The PVL Module is delimited by either the end of the provided Octet Sequence or by the use of an optional End Statement (see 2.5).

Figure 2-1 contains a syntax diagram for the contents of the PVL Module; it references Figure 2-2, which defines WSC to represent a possibly empty collection of White Space Characters and/or Comments. When this construct appears in syntax diagrams, it represents the capability of using optional White Space and/or Comments between syntactic elements for readability.

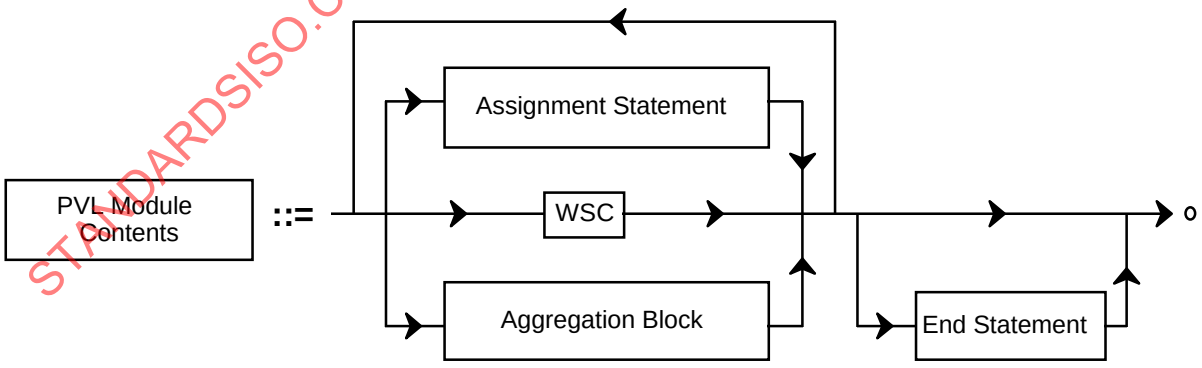


Figure 2-1: PVL Module Contents Syntax Diagram

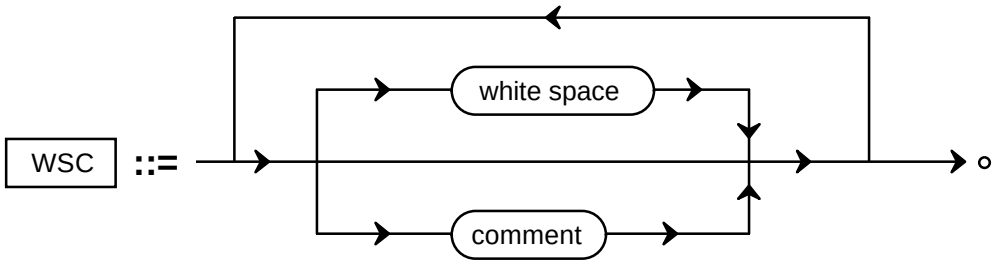


Figure 2-2: White Space/Comment Syntax Diagram

An Assignment Statement has the following general form:

Parameter = Value

An Aggregation Block has the following general form:

Begin Aggregation Statement

A collection of Assignment Statements and/or Aggregation Blocks

End Aggregation Statement

2.3 ASSIGNMENT STATEMENT

An Assignment Statement is used to assign a value to a Parameter Name. Within the Assignment Statement, White Spaces and Comments are ignored between syntactic elements, except where required for statement delimitation.

The Assignment Statement has the following format (the square brackets indicate that the semicolon is optional):

parameter name = value [;]

Figure 2-3 contains a syntax diagram for the PVL Assignment Statement.

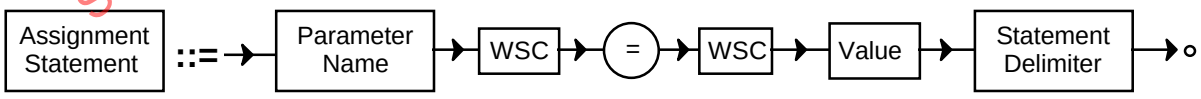


Figure 2-3: Assignment Statement Syntax Diagram

Figure 2-4 contains a syntax diagram illustrating the options for Statement Delimiter.

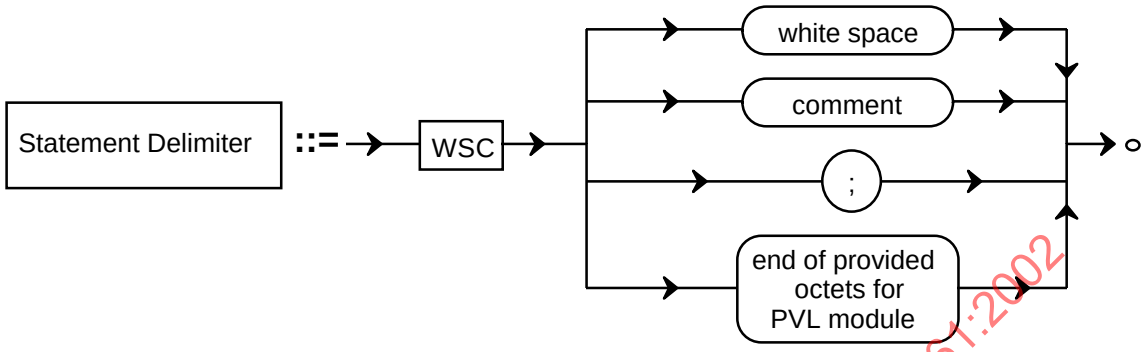


Figure 2-4: Statement Delimiter Syntax Diagram

Statements are separated by the use of a Statement Delimiter, which follows the value. Within this context, a Statement Delimiter is defined as one of the following:

– an explicit delimiter character (`;`), which can be preceded by White Space Characters and/or Comments;

– in the absence of the explicit delimiter character, a set of one or more White Space Characters or Comments;

– the end of the externally provided octet sequence.

Statement delimitation in the absence of the explicit delimiter character is the only time when White Space or Comments have semantic meaning in PVL.

NOTE – Since any of the Statement Delimiters are syntactically equivalent, the use of a particular Statement Delimiter may not be used to provide different meanings (semantics) for applications.

2.3.1 PARAMETER NAME

The Parameter Name provides a way to reference the value assigned in the Assignment Statement. Parameter Names consist of a sequence of Unrestricted Characters.

A Parameter Name must not contain a Comment Delimiter sequence (`/*` or `*/`) and it shall not conform to Numeric encoding rules (see 2.3.2.1.1) or Date/Time encoding rules (see 2.3.2.1.3). A Parameter Name terminates with the character immediately prior to a Reserved Character, a White Space Character, or the beginning of a Comment.

There are seven reserved keywords in PVL:

```
BEGIN_GROUP
BEGIN_OBJECT
END
END_GROUP
END_OBJECT
GROUP
OBJECT
```

Reserved keywords are not permitted as Parameter Names within an Assignment Statement or as Block Names in Aggregation Statements.

2.3.2 VALUE

The Value in an Assignment Statement can be a Simple Value, a Set, or a Sequence. Any Simple Value, Set, or Sequence can optionally be followed by a Units Expression.

Figure 2-5 contains a syntax diagram for Value.

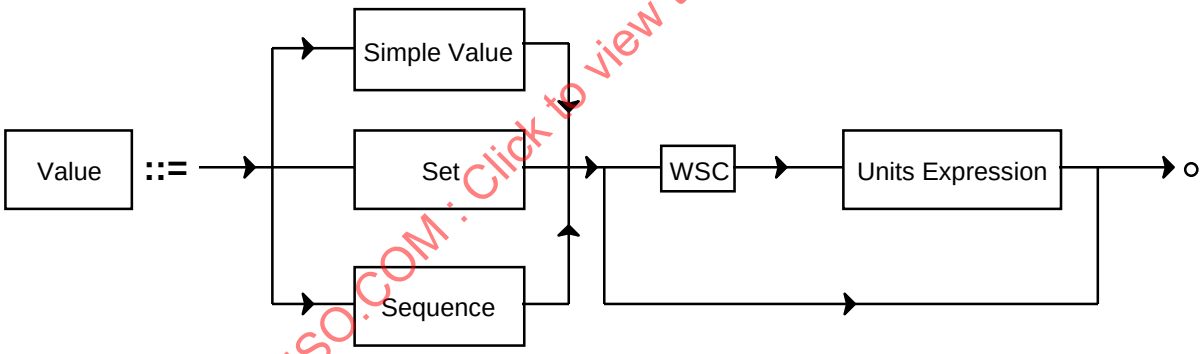


Figure 2-5: Value Syntax Diagram

2.3.2.1 Simple Value

A Simple Value can be a Numeric, String, or Date/Time Value.

Figure 2-6 contains a syntax diagram for a Simple Value.

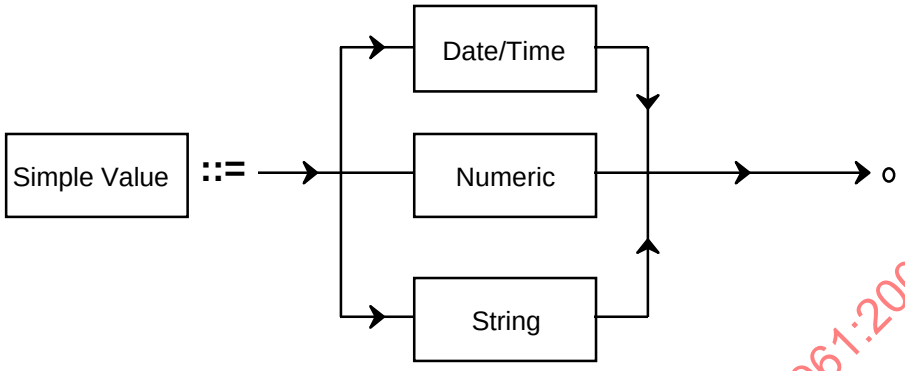


Figure 2-6: Simple Value Syntax Diagram

2.3.2.1.1 Numeric

A Numeric is a sequence of Unrestricted Characters that conform to encoding rules that permit its interpretation as a number. Numerics can be either Decimal Numbers or one of three non-decimal integer encodings: Binary, Octal, and Hexadecimal.

2.3.2.1.1.1 Decimal Numbers

Decimal Numbers follow the three numerical representations (Integer, Floating Point, and Exponential) specified in ISO 6093 (see Reference [3]) for decimal representations, with the exception that comma (,) shall not be used as a decimal point.

2.3.2.1.1.1.1 Integer

Integer numbers correspond to the First Numerical Representation (NR1) in ISO 6093. Each number is represented by at least one decimal digit. The number can be optionally prefixed by a sign symbol (+ or -). An unsigned number will be taken as positive.

Examples:
```
125
+211109
-79
```

2.3.2.1.1.1.2 Floating Point

Floating Point numbers correspond to the Second Numerical Representation (NR2) in ISO 6093. Each number is represented by at least one decimal digit and a decimal point. The decimal point is defined to be the full stop (.). The decimal point can appear anywhere within the sequence. The decimal point is used to separate the integer part of the real number from the

fractional part, at the point where the decimal point is placed. The number can be optionally prefixed by a sign symbol (`+` or `-`). An unsigned number will be taken as positive.

Examples:
```
    69.35
+12456.345
   -0.23456
     .05
   -7.
```

2.3.2.1.1.1.3 Exponential

Exponential numbers correspond to the Third Numerical Representation (NR3) in ISO 6093. Each number is represented by two sequences of decimal digits called the significand (i.e., mantissa) and exponent, separated by the ASCII character `E` or `e`. The value of the number equals the value of the significand multiplied by the result of 10 raised to the power represented by the exponent. The significand can be optionally prefixed by the sign symbol (`+` or `-`). The exponent is an optionally signed integer. If either the significand or exponent is unsigned, it will be taken as positive.

Examples:
```
-2.345678E12
   1.567E-10
   +4.99E+3
```

2.3.2.1.1.2 Non-Decimal Representations

Integer values can also be represented in bases other than base 10. Non-decimal Integers can have a radix of `2` (binary), `8` (octal), or `16` (hexadecimal). The Non-decimal Integer format begins with an optional sign (`+` or `-`) and the radix (in decimal notation), followed by the non-decimal form enclosed within a pair of number signs (`#`). If the optional leading sign has been omitted, then the number will be taken as positive. The non-decimal form itself is interpreted as a positive, uncomplemented integer.

A Non-decimal Integer has the following form

[sign]radix#non_decimal_integer#

where the radix denotes whether the number is binary, octal, or hexadecimal.

NOTE – Any of the above forms of Numerics are stored internally by PVL support software in a format that may be unknown to the user. Therefore, if a particular string of bits is required or must be conserved, for instance, as a mask or flag, then this should be expressed as a Quoted String, e.g. `MASK = "01110101";` and translated to a bit pattern by the application.

2.3.2.1.1.2.1 Binary Numbers

Binary Numbers are represented with a radix value of `2`. The non-decimal portion is a sequence of the characters `0` or `1`.

Example: *`2#0101#`* is equal to the decimal value 5.

2.3.2.1.1.2.2 Octal Numbers

Octal Numbers are represented with a radix value of `8`. The non-decimal portion is a sequence of characters from the following set:

`0, 1, 2, 3, 4, 5, 6, 7.`

Example: *`8#0107#`* is equal to decimal value 71.

2.3.2.1.1.2.3 Hexadecimal Numbers

Hexadecimal Numbers are represented with a radix value of `16`. The non-decimal portion is a sequence of characters from the following set:

`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, a, b, c, d, e, f`

Lower case letters are equivalent to their upper case counterparts.

Example: *`16#100A#`* is equal to the decimal value 4106.

2.3.2.1.2 String

A String is a sequence of PVL Characters that conforms to the requirements for either a Quoted String or an Unquoted String. Figure 2-7 contains a syntax diagram illustrating the two types of String.

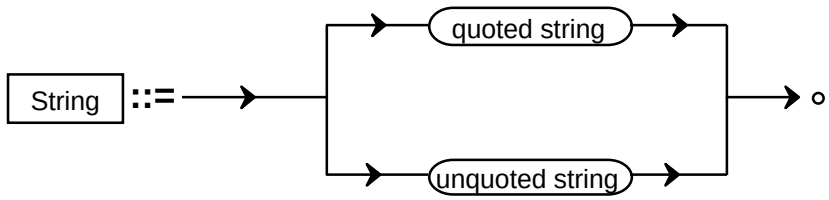


Figure 2-7: String Type Syntax Diagram

2.3.2.1.2.1 Quoted String

A Quoted String consists of zero or more PVL Characters enclosed between matching quote delimiters. The Quote String Delimiters are the quotation mark (") or the apostrophe (').

NOTES

1 A Quote String Delimiter character can be embedded within a String by the use of the Quote String Delimiter not used to enclose the String itself (e.g. "John said 'GOODBYE' and then left" or 'John said "GOODBYE" and then left').

2 If a String is to contain any of the Reserved Characters, White Space Characters, or the Comment Delimiter sequences, it must be a Quoted String rather than an Unquoted String. A String must also be quoted if it conforms to the encoding rules for either Numeric or Date/Time.

3 The above definition allows for null length (i.e., empty) Strings. A null length String may have meaning and therefore is permitted.

4 Since Quote String Delimiters are interchangeable, the use of a particular Quote String Delimiter may not be used to provide different meanings (semantics) for applications.

2.3.2.1.2.2 Unquoted String

An Unquoted String is a sequence of one or more Unrestricted Characters.

An Unquoted String shall not contain the Begin Comment Delimiter (/*) or the End Comment Delimiter (*/), nor shall it conform to Numeric or Date/Time encoding rules. An Unquoted String terminates with the character immediately prior to a White Space Character, a Reserved Character, the beginning of a Comment, or the end of the PVL Module.

2.3.2.1.3 Date/Time Value

The Date/Time Value is a strict subset of the CCSDS ASCII Time Code recommendation (Reference [3]), in which all time is represented in Universal Coordinated Time, (i.e. Greenwich Mean Time). The time construct consists of a combination of date and time constructs.

The date construct has two forms:

YYYY-DDD
where
YYYY is year (0001 to 9999)
DDD is day of year (001 to 365, 366 for leap year)

and

YYYY-MM-DD
where
YYYY is four digit year (0001 to 9999)
MM is month (01 to 12)
DD is day of month (01 to 28, 29, 30 or 31)

Note that each field has a specified width, leading zeros must be included if needed to assure field width. Figure 2-8 contains a syntax diagram of the date format.

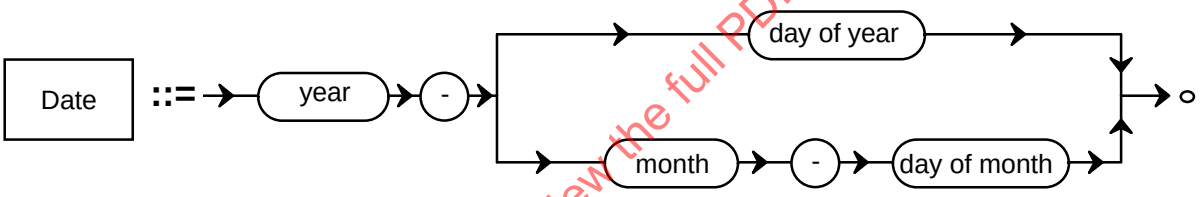


Figure 2-8: Date Syntax Diagram

Examples: `2000-012` is the twelfth day of the year 2000
`1995-06-08` is June 8, 1995
`1978-04-30` is April 30, 1978

The time construct has the form

hh:mm[:ss[.d...d]]
where
hh is hours (00 to 23)
mm is minutes (00 to 59)
ss is seconds (00 to 60, 60 is to accommodate leap seconds).
d...d is fractional seconds represented by 1 or more digits.

Figure 2-9 contains a syntax diagram for the time format.

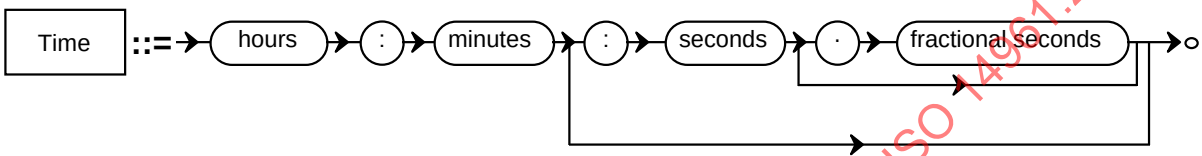


Figure 2-9: Time Syntax Diagram

Examples: `00:00:00.0`
`12:01:56`
`23:01`

The complete time construct consists of date, followed by the separator `T` followed by the time construct; all of this can be optionally followed by the character `Z` as a terminator. Separate time and date values can also be used. Figure 2-10 contains a syntax diagram of the Date/Time format.

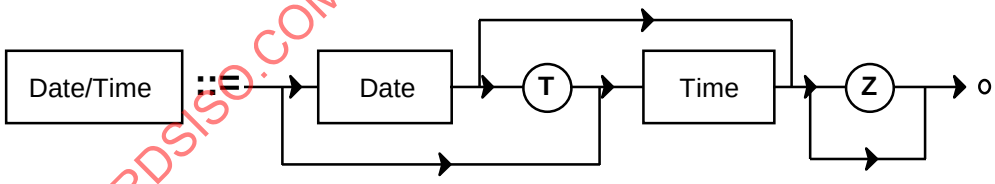


Figure 2-10: Date/Time Syntax Diagram

Examples: `1991-12-22T22:03:12.01Z`
`2001-001T12:13`
`1998-02-12T00:00:01.00`
`1995-360T14:02:13.0123456Z`

2.3.2.2 Set

A Set is a delimited collection of Values in which the order of the Values is not significant and need not be maintained. A Set can contain zero or more Values. If a Set contains two or more Values, they are separated by commas. The beginning of a Set is indicated by a left curly bracket ({), and the end by a right curly bracket (}).

NOTE – The above definition allows for Empty Sets. An Empty Set may have meaning and is therefore permitted.

Figure 2-11 contains a syntax diagram for the Set format.

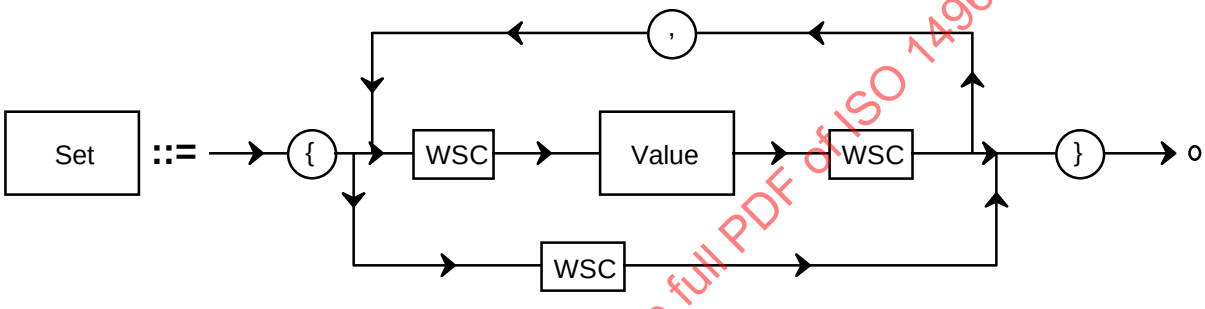


Figure 2-11: Set Syntax Diagram

2.3.2.3 Sequence

A Sequence is a delimited collection of Values in which the order of the Values is significant. A Sequence can contain zero or more Values. If two or more Values are contained in a Sequence, they are separated by commas. The beginning of a Sequence is indicated by a left parenthesis (( ) and the end by a right parenthesis ( )).

NOTE – The above definition allows for Empty Sequences. An Empty Sequence may have meaning and is therefore permitted.

Figure 2-12 contains a syntax diagram for the Sequence format.

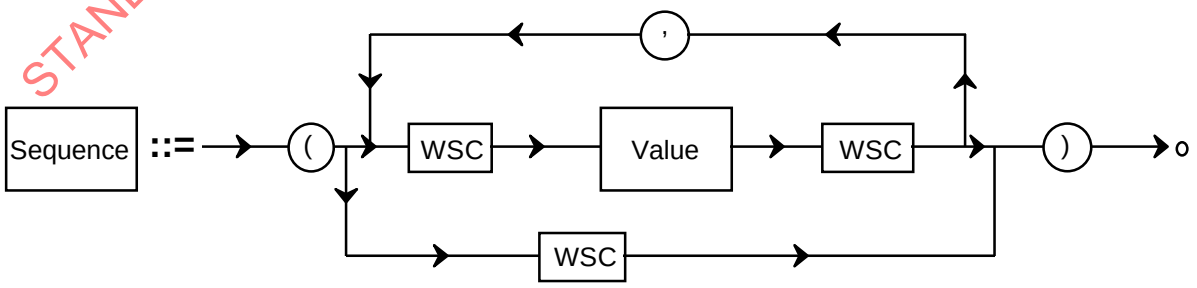


Figure 2-12: Sequence Syntax Diagram

2.3.2.4 Units Expression

Any Simple Value, Set, or Sequence can optionally be followed by a Units Expression. The Units Expression consists of a Units Value contained between an open angle bracket (<) and a close angle bracket (>). The Units Value begins with the first non–White Space Character after the open angle bracket and ends with the last non–White Space Character before the close angle bracket. A Units Value can contain any PVL Character other than the angle brackets themselves.

Figure 2-13 contains a syntax diagram for the Units Expression format. Note that White Space is a collection of one or more White Space Characters.

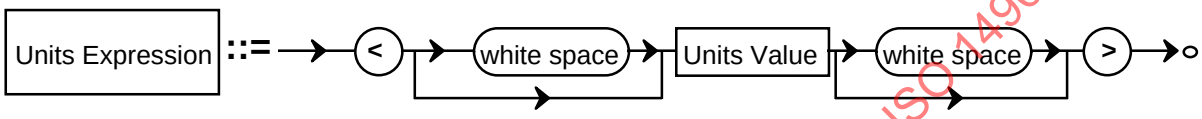


Figure 2-13: Units Expression Syntax Diagram

Figure 2-14 contains a syntax diagram for the Units Value format, in which a units character is any PVL Character other than the open angle bracket (<), close angle bracket (>), or White Space Character.

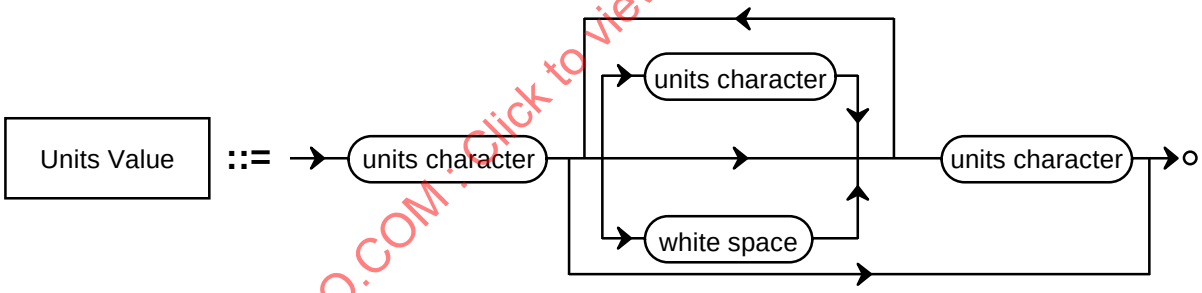


Figure 2-14: Units Value Syntax Diagram

2.4 AGGREGATION BLOCK

The Aggregation Block is a named collection of Assignment Statements and/or other Aggregation Blocks. The Aggregation Block is identified by a Block Name. The start of the block is indicated by a Begin Aggregation Statement and is terminated by an End Aggregation Statement. Figure 2-15 contains a syntax diagram for the Aggregation Block format.

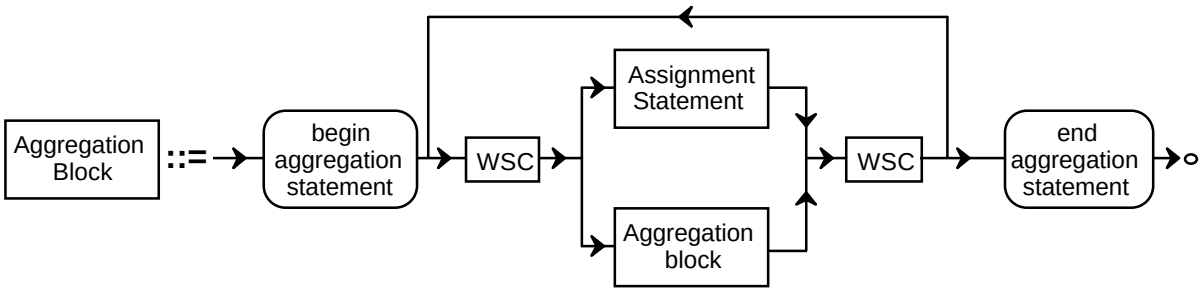


Figure 2-15: Aggregation Block Syntax Diagram

Aggregations are commonly referred to as Groups or Objects. These two keyword forms for Aggregation Statements are permitted to allow for the stylistic preferences. No semantic differentiation between the two is made by PVL. Applications are free to assign such differentiation if desired.

2.4.1 BEGIN AGGREGATION STATEMENT

The Begin Aggregation Statement is parallel in construction to the Assignment Statement. The Begin Aggregation Statement has the following format (the square brackets indicate that the semicolon is optional):

begin aggregation keyword = block name [;]

The Begin Aggregation keywords are `BEGIN_GROUP` and `BEGIN_OBJECT` and are matched with statements that use `END_GROUP` and `END_OBJECT` respectively. The keyword `OBJECT` is a synonym for `BEGIN_OBJECT` and the keyword `GROUP` is a synonym for `BEGIN_GROUP`. The form of the Block Name is identical to Parameter Name.

NOTES

1 These synonyms are allowed for historical compatibility with several existing keyword languages.

2 Since BEGIN_GROUP and GROUP are syntactically equivalent and BEGIN_OBJECT and OBJECT are syntactically equivalent, the use of synonymous forms of the keyword may not be used to provide different meanings (semantics) for applications.

3 Since BEGIN_GROUP and BEGIN_OBJECT are **not** syntactically equivalent, applications may assign different meanings (semantics) to their use.

Figure 2-16 contains a syntax diagram of the Begin Aggregation Statement formats.

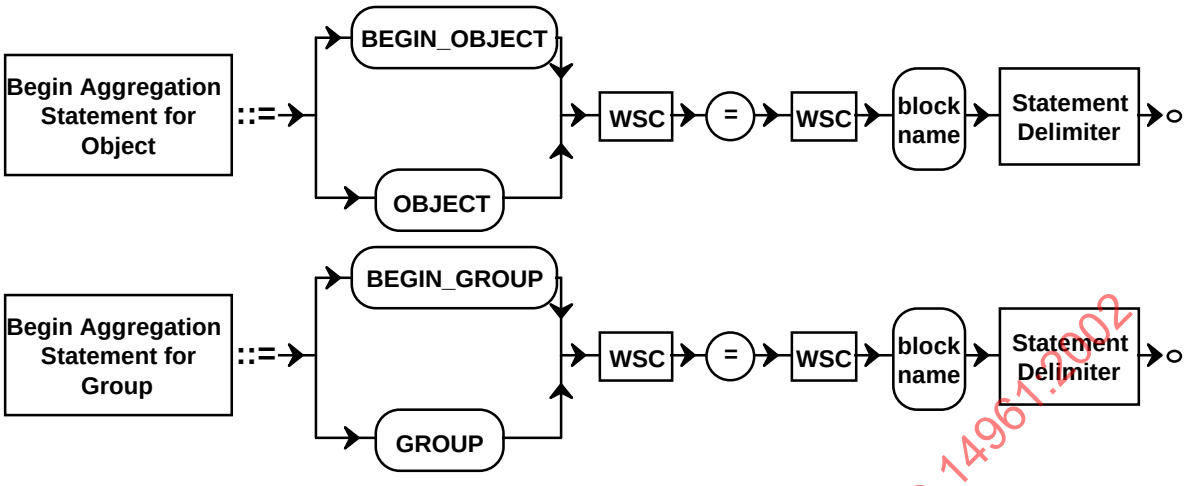


Figure 2-16: Aggregation Begin Statement Syntax Diagram

2.4.2 END AGGREGATION STATEMENT

The End Aggregation Statement is identified by the End Aggregation keyword. The full form of the End Aggregation Statement follows the same construction rules as an Assignment Statement; it has the following format (the square brackets indicate that the semicolon is optional):

end aggregation keyword = Block Name [;]

An abbreviated form of the End Aggregation Statement is allowed as a convenience to the user. The abbreviated End Aggregation Statement has the following format:

end aggregation keyword [;]

The use of the full form is encouraged.

NOTE – Since the full form and the abbreviated form are syntactically equivalent, the use of the abbreviated form rather than the full form may not be used to provide different meanings (semantics) for applications.

The defined End Aggregation keywords are `END_GROUP` and `END_OBJECT`. Figure 2-17 contains a syntax diagram for the End Aggregation Statement.

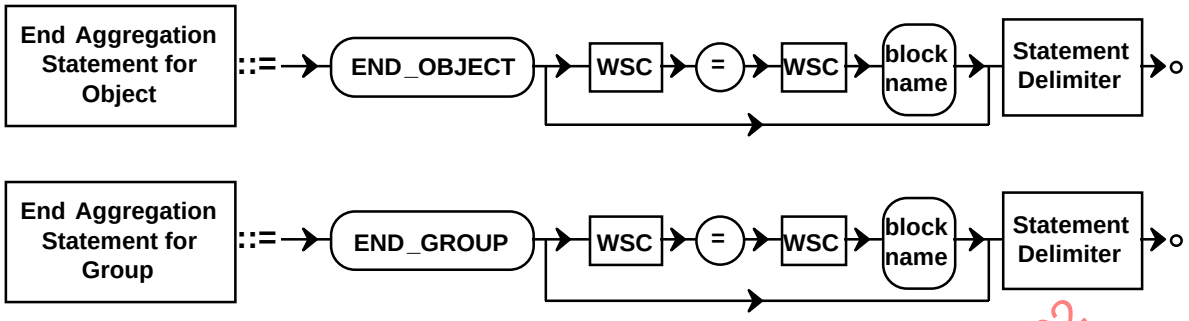


Figure 2-17: End Aggregation Statement Syntax Diagram

NOTE – The preferred form of the aggregation end statement is the full form, which includes the Block Name.

2.4.3 AGGREGATION BLOCK CONSTRUCTION RULES

The end aggregation statement must be paired with a begin aggregation statement. In other words, an Aggregation Block that starts with a `BEGIN_GROUP` statement must end with an `END_GROUP` statement. If a Block Name is used in the end aggregation statement, it must match the name used in the matching begin aggregation statement.

2.5 END STATEMENT

The End Statement is a special type of statement used to delimit a PVL Module prior to the end of the externally provided octet sequence. Figure 2-18 illustrates the syntax

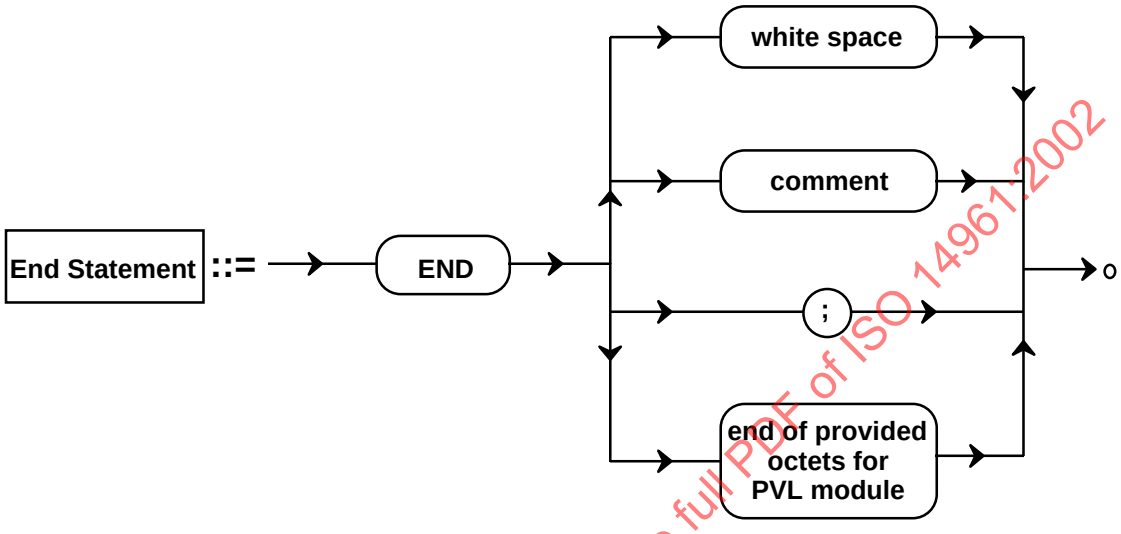


Figure 2-18: End Statement Syntax Diagram

The End Statement is delimited by one of the following: a semicolon; the first White Space Character; the end of a Comment; or the end of the provided octet space.

NOTE – The statement delimitation of the End Statement is more restrictive than for other statements since the remaining Octets in the sequence which may include White Space, Comments, or semicolons, as well as any other character, may have significance to the application.

There shall be at most one End Statement in a PVL Module, and if present it shall be the last statement of the PVL Module.

3 CONSTRAINTS FOR INFORMATION PRESERVATION

To ensure that information is preserved in the exchange of PVL objects among open systems, it is necessary to make clear which PVL formatting options may and may not be altered by these systems. Producing systems will then avoid attaching special meaning to formatting choices that may be altered by automated processes in recipient systems.

The following constraints need to be observed:

1. Statement ordering shall be preserved.

NOTE – Specific applications built on PVL are free to allow statement re-ordering as long as aggregations (`BEGIN_GROUP` and `BEGIN _OBJECT`) are correctly preserved.

2. Statement Delimiters (e.g., White Space, semicolons) may be substituted for each other.

NOTE – The use of a semicolon as the Statement Delimiter is the preferred form.

3. Comments may be added or deleted.

NOTE – Maintaining Comments is the preferred form. Additional Comments may be added as is consistent with valid PVL.

4. White Space between PVL statement elements may be altered as to amount and type.

5. String Delimiters may be added to PVL Strings or removed from PVL Strings as is consistent with valid PVL and where the meaning of the String is not changed. For example, the string "ABCD1234" (with double quotes) may be represented as ‘ABCD1234’ (with single quotes) or as ABCD1234 (without quotes), and vice-versa.

6. PVL Begin Aggregation keywords `GROUP` and `BEGIN_GROUP` may be substituted for each other.

NOTE – The use of the `BEGIN_GROUP` keyword is the preferred form.

7. PVL begin aggregation keywords `OBJECT` and `BEGIN_OBJECT` may be substituted for each other.

NOTE – The use of the `BEGIN_OBJECT` keyword is the preferred form.

8. PVL end aggregation statements `END_GROUP = 'Block Name';` and `END_GROUP;` may be substituted for each other.

NOTE – `END_GROUP = 'Block Name';` is the preferred form.

9. PVL end aggregation statements `END_OBJECT = 'Block Name';` and `END_OBJECT;` may be substituted for each other.

NOTE – `END_OBJECT = 'Block Name';` is the preferred form.

(Blank page)

4 PARAMETER VALUE LANGUAGE FORMAL SYNTAX SPECIFICATION

Precedence: In the case of ambiguity of the preceding sections or disagreement with this formal specification, this formal specification shall take precedence. This specification is presented in Abstract Syntax Notation One (ASN.1, see Reference[2]). The comments in the ASN.1 are also part of the specification. Readers unfamiliar with ASN.1 may wish to consult an ASN.1 tutorial such as Reference [C4].

The ASN.1 specification is organized into groupings based on major constructs. Each group begins on a new page with Comment block immediately followed by the definition of the construct with its components in alphabetical order. Components used by more than one major construct are listed in the common language elements group at the end of the specification. Common language elements contain components such as Statement Delimiter, separator, the combination of White Space and Comment (WSC), and character sets.

The construct sections are found on the following pages:

NOTE – The term IA5String as used in this ASN.1 refers to the International ASCII Character Set #5.

4.1 FORMAL SPECIFICATION

```
PVLModule DEFINITIONS      ::= BEGIN
```

CCSDS RECOMMENDATION FOR PARAMETER VALUE LANGUAGE SPECIFICATION

```
-- ********************************************************
-- *****
-- *****   PVL MODULE CONTENTS
-- *****
-- ********************************************************

PVLModuleContents       ::= SEQUENCE
                                {
                                SEQUENCE OF
                                    CHOICE
                                        {
                                        Statement,
                                        WSC
                                        },
                                EndStatement OPTIONAL
                                }

EndKeyword              ::= IA5String("END")

EndStatement            ::= SEQUENCE
                                {
                                EndKeyword,
                                CHOICE
                                    {
                                    SemiColon,
                                    WhiteSpace,
                                    Comment,
                                    EndProvidedOctetSeq
                                    }
                                }

Statement               ::=  CHOICE
                                {
                                AssignmentStmt,
                                AggregationBlock
                                }
```

```
-- ********************************************************
-- *****
-- *****   AGGREGATION BLOCK
-- *****
-- ********************************************************

AggregationBlock        ::= CHOICE
                            {
                            AggrGroup,
                            AggrObject
                            }

AggrContents            ::= SEQUENCE
                            {
                            WSC,
                            Statement,      -- Must contain at least one statement
                            SEQUENCE OF
                                CHOICE
                                {
                                WSC,
                                Statement
                                }
                            }

AggrGroup               ::= SEQUENCE
                            {
                            BeginGroupStmt,
                            AggrContents,
                            EndGroupStmt
                            }

AggrObject              ::= SEQUENCE
                            {
                            BeginObjectStmt,
                            AggrContents,
                            EndObjectStmt
                            }
```

```
BeginGroupKeywd         ::= CHOICE
                            {
                            IA5String("BEGIN_GROUP"),
                            IA5String("GROUP")
                            }

BeginGroupStmt          ::= SEQUENCE
                            {
                            BeginGroupKeywd,
                            WSC,
                            AssignmentSymbol,
                            WSC,
                            BlockName,
                            -- Block Name must match Block Name
                            -- in paired End Group Statement
                            -- if Block Name is present in End Group Statement
                            StatementDelim
                            }

BeginObjectKeywd        ::= CHOICE
                            {
                            IA5String("BEGIN_OBJECT"),
                            IA5String("OBJECT")
                            }

BeginObjectStmt         ::= SEQUENCE
                            {
                            BeginObjectKeywd,
                            WSC,
                            AssignmentSymbol,
                            WSC,
                            BlockName,
                            -- Block Name must match Block Name
                            -- in paired End Object Statement
                            -- if Block Name is present in End Object Statement
                            StatementDelim
                            }
```

```
BlockName           ::= SEQUENCE
                        {
                        -- Must not contain the sequence /* or */
                        -- Must not be reserved keyword (see 4.2)
                        -- Must not conform to Numeric encoding rules
                        -- Must not conform to Date/Time encoding rules
                        UnrestrictedChar,
                        SEQUENCE OF UnrestrictedChar
                        }

EndGroupKeywd       ::= IA5String("END_GROUP")

EndGroupLabel       ::= SEQUENCE
                        {
                        AssignmentSymbol,
                        WSC,
                        BlockName
                        -- Block Name must match Block Name
                        -- in paired Begin Group Statement
                        -- if End Group Label is present
                        -- in End Group Statement
                        }

EndGroupStmt        ::= SEQUENCE
                        {
                        EndGroupKeywd,
                        WSC,
                        EndGroupLabel OPTIONAL,
                        StatementDelim
                        }
```

```
EndObjectLabel          ::= SEQUENCE
                            {
                            AssignmentSymbol,
                            WSC,
                            BlockName
                            -- Block Name must match Block Name
                            -- in paired Begin Object Statement
                            -- if End Object Label is present
                            -- in End Object Statement
                            }

EndObjectKeywd          ::= IA5String("END_OBJECT")

EndObjectStmt           ::= SEQUENCE
                            {
                            EndObjectKeywd,
                            WSC,
                            EndObjectLabel OPTIONAL,
                            StatementDelim
                            }
```

```
-- ********************************************************
-- *****
-- *****   ASSIGNMENT STATEMENT
-- *****
-- ********************************************************

AssignmentStmt          ::= SEQUENCE
                            {
                            Name,
                            WSC,
                            AssignmentSymbol,
                            WSC,
                            Value,
                            StatementDelim
                            }

Name                    ::= SEQUENCE
                            -- Must not contain the sequence /* or */
                            -- Must not be reserved keyword (see 4.2)
                            -- Must not conform to Numeric encoding rules
                            -- Must not conform to Date/Time encoding rules
                            {
                            UnrestrictedChar,
                            SEQUENCE OF UnrestrictedChar
                            }

SimpleValue             ::= CHOICE
                            {
                            Numeric,
                            String,
                            DateTimeValue
                            }

Value                   ::= SEQUENCE
                            {
                            CHOICE
                                {
                                SimpleValue,
                                Set,
                                Sequence
                                },
                            WSC,
                            UnitsExpression OPTIONAL
                            }
```

```
-- ********************************************************
-- *****
-- *****   COMMENT
-- *****
-- ********************************************************

Comment             ::= SEQUENCE
                        {
                        CommentStart,
                        CommentString,
                        CommentEnd
                        }

CommentChar         ::= CHOICE
                        {
                        UnrestrictedChar,
                        WhiteSpace,
                        Apostrophe,
                        QuoteMark,
                        OpenAngleBracket,
                        CloseAngleBracket,
                        SpecialChar
                        }

CommentEnd          ::= IA5String("*/")

CommentStart        ::= IA5String("/*")

CommentString       ::= SEQUENCE OF CommentChar
                        -- Must not contain the sequence "/*" or "*/"
```

```
-- ********************************************************
-- *****
-- *****   DATE/TIME VALUE
-- *****
-- ********************************************************

DateTimeValue           ::= SEQUENCE
                            {
                            CHOICE
                                {
                                Date,
                                Time,
                                SEQUENCE
                                    {
                                    Date,
                                    DateTimeSeparator,
                                    Time
                                    }
                                },
                            TimeCodeTerminator OPTIONAL
                            }

Colon                   ::= IA5String(":")

Date                    ::= SEQUENCE
                            {
                            Year,
                            Hyphen,
                            CHOICE
                                {
                                DayOfYear,
                                MonthAndDay
                                }
                            }
DateTimeSeparator       ::= IA5String("T")
```

```
DayOfMonth          ::= CHOICE
                        {
                        SEQUENCE
                            {
                            -- 01 to 09
                            DecimalChar0,
                            PosDecimalChar
                            },
                        SEQUENCE
                            {
                            -- 10 to 29
                            DecimalChar1to2,
                            DecimalChar
                            },
                        SEQUENCE
                            {
                            -- 30 to 31
                            DecimalChar3,
                            DecimalChar0to1
                            }
                        }
```

```
DayOfYear               ::= CHOICE
                            {
                            SEQUENCE
                                {
                                -- 001 to 009
                                DecimalChar0,
                                DecimalChar0,
                                PosDecimalChar
                                },
                            SEQUENCE
                                {
                                -- 010 to 099
                                DecimalChar0,
                                PosDecimalChar,
                                DecimalChar
                                },
                            SEQUENCE
                                {
                                -- 100 to 299
                                DecimalChar1to2,
                                DecimalChar,
                                DecimalChar
                                },
                            SEQUENCE
                                {
                                -- 300 to 366
                                DecimalChar3,
                                CHOICE
                                    {
                                    SEQUENCE
                                        {
                                        -- 300 to 359
                                        DecimalChar0to5,
                                        DecimalChar
                                        },
                                    SEQUENCE
                                        {
                                        -- 360 to 366
                                        DecimalChar6,
                                        DecimalChar0to6
                                        }
                                    }
                                }
                            }
```

```
DecFracSecond           ::= SEQUENCE
                                {
                                DecimalChar,
                                SEQUENCE OF DecimalChar
                                }

DecFracSecondSeq        ::= SEQUENCE
                                {
                                DecimalPoint,
                                DecFracSecond
                                }

DecimalChar0            ::= IA5String("0")

DecimalChar0to1         ::= IA5String(FROM ("0" | "1"))

DecimalChar0to2         ::= IA5String(FROM ("0"| "1"| "2"))

DecimalChar0to3         ::= IA5String(FROM ("0"| "1"| "2"| "3"))

DecimalChar0to5         ::= IA5String(FROM ("0" | "1"| "2"| "3"| "4"| "5"))

DecimalChar0to6         ::= IA5String(FROM ("0" | "1"| "2"| "3"| "4"| "5" |" 6"))

DecimalChar1            ::= IA5String("1")

DecimalChar1to2         ::= IA5String(FROM ("1"| "2"))

DecimalChar2            ::= IA5String("2")

DecimalChar3            ::= IA5String("3")

DecimalChar6            ::= IA5String("6")

DecimalChar60           ::= IA5String("60")
```

```
Hour            ::= CHOICE
                    {
                    SEQUENCE
                        {
                        -- 00 to 19
                        DecimalChar0to1,
                        DecimalChar
                        },
                    SEQUENCE
                        {
                        -- 20 to 23
                        DecimalChar2,
                        DecimalChar0to3
                        }
                    }

Hyphen          ::= IA5String("-")

Minute          ::= SEQUENCE
                    {
                    -- 00 to 59
                    DecimalChar0to5,
                    DecimalChar
                    }

Month           ::= CHOICE
                    {
                    SEQUENCE
                        {
                        -- 00 to 09
                        DecimalChar0,
                        PosDecimalChar
                        },
                    SEQUENCE
                        {
                        -- 10 to 12
                        DecimalChar1,
                        DecimalChar0to2
                        }
                    }
```

```
MonthAndDay             ::= SEQUENCE
                            {
                            Month,
                            Hyphen,
                            DayOfMonth
                            }

PosDecimalChar          ::= IA5String(FROM ("1"| "2"| "3"| "4"| "5"| "6"| "7"| "8"| "9"))

Second                  ::= CHOICE
                            {
                            SEQUENCE
                                {
                                -- 00 to 59
                                DecimalChar0to5,
                                DecimalChar
                                },
                            DecimalChar60
                            --  60 is allowed for leap seconds
                            }

SecondSeq               ::= SEQUENCE
                            {
                            Colon,
                            Second,
                            DecFracSecondSeq OPTIONAL
                            }

Time                    ::= SEQUENCE
                            {
                            Hour,
                            Colon,
                            Minute,
                            SecondSeq OPTIONAL
                            }

TimeCodeTerminator      ::= IA5String("Z")

Year                    ::= SEQUENCE
                            {
                            -- year 0000 is not allowed --
                            DecimalChar,
                            DecimalChar,
                            DecimalChar,
                            DecimalChar
                            }
```

```
-- ********************************************************
-- *****
-- *****   NUMERIC VALUES
-- *****
-- ********************************************************

Numeric             ::= CHOICE
                        {
                        Integer,
                        FloatingPoint,
                        Exponential,
                        BinaryNum,
                        OctalNum,
                        HexadecimalNum
                        }

BinaryChar          ::= IA5String(FROM("0" |"1"))

BinaryNum           ::= SEQUENCE
                        {
                        Sign OPTIONAL,
                        IA5String("2"),
                        RadixSymbol,
                        -- Binary characters are interpreted
                        -- as a positive and uncomplemented integer
                        BinaryChar,
                        SEQUENCE OF BinaryChar,
                        RadixSymbol
                        }
```

```
Exponential          ::= SEQUENCE
                         {
                         CHOICE
                             {
                             Integer,
                             FloatingPoint
                             },
                         ExponentMark,
                         Integer
                         }

ExponentMark         ::= IA5String(FROM("e" |"E"))

FloatingPoint        ::= SEQUENCE
                         {
                         Sign OPTIONAL,
                         -- If all digits in number are 0,
                         --   only legal value for sign is +
                         CHOICE
                             {
                             SEQUENCE
                                 {
                                 DecimalChar,
                                 -- Ensures at least one digit to
                                 --  left of decimal point
                                 SEQUENCE OF DecimalChar,
                                 DecimalPoint,
                                 SEQUENCE OF DecimalChar
                                 },
                             SEQUENCE
                                 {
                                 SEQUENCE OF DecimalChar,
                                 DecimalPoint,
                                 DecimalChar,
                                 -- Ensures at least one digit to
                                 --  right of decimal point
                                 SEQUENCE OF DecimalChar
                                 }
                             }
                         }

HexadecimalChar      ::= IA5String(FROM("0" |"1" |"2" |"3" |"4" |"5" |"6" |"7" |"8"
                             |"9" |"A" |"B" |"C" |"D" |"E" |"F" |"a" |"b" |"c" |"d"
                             |"e" |"f"))
```

```
HexadecimalNum      ::= SEQUENCE
                        {
                        Sign OPTIONAL,
                        IA5String("16"),
                        RadixSymbol,
                        -- Hexadecimal characters are interpreted
                        --   as a positive and uncomplemented integer
                        HexadecimalChar,
                        SEQUENCE OF HexadecimalChar,
                        RadixSymbol
                        }

Integer             ::= SEQUENCE
                        {
                        Sign OPTIONAL,
                        -- If all digits in number are 0,
                        --   only legal value for sign is +
                        DecimalChar,
                        SEQUENCE OF DecimalChar
                        }

OctalChar           ::= IA5String(FROM("0" |"1" |"2" |"3" |"4" |"5" |"6" |"7"))

OctalNum            ::= SEQUENCE
                        {
                        Sign OPTIONAL,
                        IA5String("8"),
                        RadixSymbol,
                        -- Octal characters are interpreted
                        --   as a positive and uncomplemented integer
                        OctalChar,
                        SEQUENCE OF OctalChar,
                        RadixSymbol
                        }

RadixSymbol         ::= IA5String("#")

Sign                ::= IA5String(FROM("+" |"-"))
```

```
-- ********************************************************
-- *****
-- *****   SEQUENCE
-- *****
-- ********************************************************

Sequence            ::= SEQUENCE
                        {
                        SequenceStart,
                        WSC,
                        SequenceValue OPTIONAL,
                        WSC,
                        SequenceEnd
                        }

SequenceEnd         ::= IA5String(")")

SequenceStart       ::= IA5String("(")

SequenceValue       ::= SEQUENCE
                        {
                        Value,
                        SEQUENCE OF
                            SEQUENCE
                                {
                                WSC,
                                SeparatorSymbol,
                                WSC,
                                Value
                                }
                        }
```

```
-- ********************************************************
-- *****
-- *****   SET
-- *****
-- ********************************************************

Set                     ::= SEQUENCE
                            {
                            SetStart,
                            WSC,
                            SetValue OPTIONAL,
                            WSC,
                            SetEnd
                            }

SetEnd                  ::= IA5String("}")

SetStart                ::= IA5String("{")

SetValue                ::= SEQUENCE
                            {
                            Value,
                            SEQUENCE OF
                                SEQUENCE
                                    {
                                    WSC,
                                    SeparatorSymbol,
                                    WSC,
                                    Value
                                    }
                            }
```

```
-- ********************************************************
-- *****
-- *****   STRING
-- *****
-- ********************************************************

String                  ::= CHOICE
                            {
                            QuotedString,
                            UnquotedString
                            }

QuotedString            ::= CHOICE
                            {
                            QuotedString1,
                            QuotedString2
                            }

QuotedChar              ::= CHOICE
                            {
                            UnrestrictedChar,
                            WhiteSpace,
                            OpenAngleBracket,
                            CloseAngleBracket,
                            SpecialChar
                            }

QstringDelim1           ::= QuoteMark

QstringDelim2           ::= Apostrophe

QuotedString1           ::= SEQUENCE
                            {
                            QstringDelim1,
                            -- quotation mark
                            SEQUENCE OF
                                CHOICE
                                    {
                                    Apostrophe,
                                    -- character used for QstringDelim2
                                    QuotedChar
                                    },
                            QstringDelim1
                            -- quotation mark
                            }
```

```
QuotedString2           ::= SEQUENCE
                                {
                                QstringDelim2,
                                -- apostrophe
                                SEQUENCE OF
                                    CHOICE
                                        {
                                        QuoteMark,
                                        -- character used for QstringDelim1
                                        QuotedChar
                                        },
                                QstringDelim2
                                -- apostrophe
                                }

UnquotedString          ::= SEQUENCE
                                -- Must not contain the sequence /* or */
                                -- Must not be reserved keyword (see 4.2)
                                -- Must not conform to Numeric encoding rules
                                -- Must not conform to Date/Time encoding rules
                                {
                                UnrestrictedChar,
                                SEQUENCE OF UnrestrictedChar
                                }
```

```
-- ********************************************************
-- *****
-- *****   UNIT EXPRESSION
-- *****
-- ********************************************************

UnitsExpression         ::= SEQUENCE
                            {
                            UnitsStart,
                            SEQUENCE OF WhiteSpace,
                            UnitsValue,
                            SEQUENCE OF WhiteSpace,
                            UnitsEnd
                            }

RemainUnitValueChars    ::= SEQUENCE
                            {
                            SEQUENCE OF
                                CHOICE
                                    {
                                    WhiteSpace,
                                    UnitsChar
                                    },
                            UnitsChar
                            }

UnitsChar               ::= CHOICE
                            {
                            UnrestrictedChar,
                            SpecialChar,
                            Apostrophe,
                            QuoteMark
                            }

UnitsEnd                ::= CloseAngleBracket

UnitsStart              ::= OpenAngleBracket

UnitsValue              ::= SEQUENCE
                            {
                            UnitsChar,
                            RemainUnitValueChars OPTIONAL
                            }
```

```
-- ********************************************************
-- *****
-- *****   COMMON LANGUAGE ELEMENTS
-- *****
-- ********************************************************

Apostrophe              ::= IA5String("'")

AssignmentSymbol        ::= IA5String(":=")

CarriageRet             ::= IA5String(13)
                            -- ASCII carriage return

CloseAngleBracket       ::= IA5String(">")

DecimalChar             ::= IA5String(FROM("0" |"1" |"2" |"3" |"4" |"5" |"6" |"7" |"8"
                                    |"9"))

DecimalPoint            ::= IA5String(".")
                            -- full stop

EndProvidedOctetSeq     ::= EXTERNAL
                            -- This is the token returned
                            -- by the system that indicates
                            -- that the end of the externally
                            -- provided Octet sequence has been reached.

AdditionalChar          ::= T61String(FROM(
    Nbsp |"¡" |"¢" |"£" |"¤" |"¥" |"¦" |"§" |"¨" |"©" |"ª" |"«" |"¬" |Shy |"®" |"¯"
    |"°" |"±" |"²" |"³" |"´" |"µ" |"¶" |"·" |"¸" |"¹" |"º" |"»" |"¼" |"½" |"¾" |"¿"
    |"À" |"Á" |"Â" |"Ã" |"Ä" |"Å" |"Æ" |"Ç" |"È" |"É" |"Ê" |"Ë" |"Ì" |"Í" |"Î" |"Ï"
    |"Ð" |"Ñ" |"Ò" |"Ó" |"Ô" |"Õ" |"Ö" |"×" |"Ø" |"Ù" |"Ú" |"Û" |"Ü" |"Ý" |"Þ" |"ß"
    |"à" |"á" |"â" |"ã" |"ä" |"å" |"æ" |"ç" |"è" |"é" |"ê" |"ë" |"ì" |"í" |"î" |"ï"
    |"ð" |"ñ" |"ò" |"ó" |"ô" |"õ" |"ö" |"÷" |"ø" |"ù" |"ú" |"û" |"ü" |"ý" |"þ" |"ÿ"))
                            -- All characters from G1 character set of ISO 8859-1

FormFeed                ::= IA5String(12)
                            -- ASCII form feed

HorizontalTab           ::= IA5String(9)
                            -- ASCII horizontal tab
```

```
Letter                  ::= IA5String(FROM("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"))

LineFeed                ::= IA5String(10)
                            -- ASCII line feed

Nbsp                    ::= T61String(160)
                            -- ISO 8859-1 nbsp character

OpenAngleBracket        ::= IA5String("<")

QuoteMark               ::= IA5String(34)
                            -- ASCII quote symbol

SemiColon               ::= IA5String(";")

SeparatorSymbol         ::= IA5String(",")

Shy                     ::= T61String(173)
                            -- ISO 8859-1 shy character

Space                   ::= IA5String(32)
                             -- ASCII space character

SpecialChar             ::= IA5String(FROM( "(" |")" |"{" | "}" |"#" |"," |";" |"=" |"[" |"]"
                                |"!" |"%" | "&" |"~" |"|" |"+" ))
                                -- Characters allowed in Comments, Quoted Strings
                                -- or Units but not in Unquoted Strings, Block Names
                                --  or Parameter Names
```

```
StatementDelim          ::= CHOICE
                            {
                            SEQUENCE
                                {
                                WSC,
                                CHOICE
                                    {
                                    SemiColon,
                                    WhiteSpace,
                                    Comment
                                    -- ensure that a Statement Delimiter consists
                                    -- of one semicolon, optionally preceded by
                                    -- multiple White Spaces and/or Comments,
                                    -- OR one or more Comments and/or
                                    -- White Space sequences.
                                    }
                                },
                            EndProvidedOctetSeq
                            }

UnrestrictedChar        ::= CHOICE
                            {
                            DecimalChar,
                            Letter,
                            AdditionalChar,  -- Included only for CCSD0008 version
                            UnrestrictedSymbol
                            }
```

```
UnrestrictedSymbol      ::= IA5String(FROM("$" |"-" |"." |"/" |":" |"?" |"@" |"\" |"^" |"_"
                                |"`" |"*" ))
VerticalTab             ::= IA5String(11)
                            -- ASCII vertical tab

WhiteSpace              ::= CHOICE
                                {
                                Space,
                                HorizontalTab,
                                VerticalTab,
                                CarriageRet,
                                LineFeed,
                                FormFeed
                                }

WSC                     ::= SEQUENCE OF
                                CHOICE
                                    {
                                    WhiteSpace,
                                    Comment
                                    }
END
```

4.2 RESERVED KEYWORDS

The following reserved keywords are not available for use as Parameter Names in Assignment Statements or as Block Names in Aggregation Statements:

```
BEGIN_GROUP
BEGIN_OBJECT
END_GROUP
END_OBJECT
END
GROUP
OBJECT
```

5 CONFORMANCE

Data conforming to a Recommendation may be said to be in conformance at some identified level. Identifying conformance levels provides a standard way to classify the required capabilities of generating and receiving systems.

This Recommendation recognizes only two conformance levels—CCSD006 Conformance and CCSD0008 Conformance.

Recipient systems that are said to be in CCSD0006 Conformance to the Recommendation shall recognize the entire specification using the CCSD0006 Character Set. Generating systems that are said to be in CCSD0006 Conformance to this Recommendation shall generate material recognizable by any CCSD0006 Conforming recipient systems.

Recipient systems that are said to be in CCSD0008 Conformance to the Recommendation shall recognize the entire specification using the CCSD0008 Character Set. Generating systems that are said to be in CCSD0008 Conformance to this Recommendation shall generate material recognizable by any CCSD0008 Conforming recipient systems.

(Blank page)