CCSDS Historical Document

This document's Historical status indicates that it is no longer current. It has either been replaced by a newer issue or withdrawn because it was deemed obsolete. Current CCSDS publications are maintained at the following location:

http://public.ccsds.org/publications/

Consultative Committee for Space Data Systems

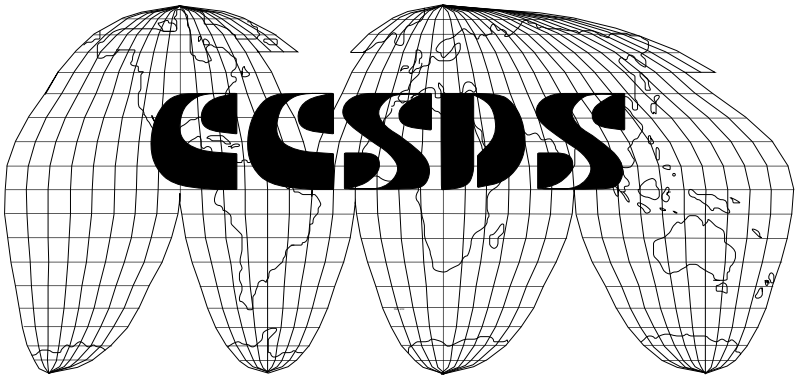
RECOMMENDATION FOR SPACE DATA SYSTEM STANDARDS

PARAMETER VALUE LANGUAGE SPECIFICATION (CCSD0006)

CCSDS 641.0-B-1

BLUE BOOK

May 1992



CCSDS HISTORICAL DOCUMENT

AUTHORITY

Issue:	Blue Book, Issue 1
Date:	May 1992
Location:	CCSDS Panel 2 Workshop Oberpfaffenhofen, Germany May 1992

This Recommendation reflects the consensus technical agreement of the following member Agencies of the Consultative Committee for Space Data Systems (CCSDS):

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

The following observer Agencies also concur with this Recommendation:

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

This Recommendation is published and maintained by:

CCSDS Secretariat
Communications and Data Systems Division, (Code-OS)
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 the date of its 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.

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 which 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 which are defined in Reference [1].

Questions relative to the contents or status of this document should be addressed to the CCSDS Secretariat.

DOCUMENT CONTROL

Document	*Title*	*Date*	*Status/ Remarks*
CCSDS 641.0-B-1	Recommendation for Space Data System Standards: Parameter Value Language, Issue 1	May 1992	Current Issue

CONTENTS

Sections

REFERENCES vii

1 INTRODUCTION 1
1.1 Purpose and Scope 1
1.2 Applicability 1
1.3 Recommended Approach to Reading the Document 1
1.4 Character Set Definitions 3
1.4.1 PVL Character Set 3
1.4.1.1 White Space Character Set 3
1.4.1.2 Reserved Character Set 3
1.4.1.3 Unrestricted Character Set 4
1.4.2 Comment 4

2 OVERVIEW OF THE LANGUAGE 5
2.1 Assignment Statement 6
2.1.1 Parameter Name 7
2.1.2 Value 7
2.1.2.1 Simple Value 8
2.1.2.1.1 Numeric 8
2.1.2.1.2 String 10
2.1.2.1.3 Date/Time Value 11
2.1.2.2 Set 13
2.1.2.3 Sequence 13
2.1.2.4 Units Expression 14
2.2 Aggregation Block 15
2.2.1 Begin Aggregation Statement 15
2.2.2 End Aggregation Statement 16
2.2.3 Aggregation Block Construction Rules 17
2.3 End Statement 18

3 PARAMETER VALUE LANGUAGE FORMAL SYNTAX SPECIFICATION 19
3.1 Formal Specification 19
3.2 Reserved Keywords 42

ANNEX A -- ACRONYMS AND GLOSSARY 43

ANNEX B -- CHARACTER DEFINITIONS 46

INDEX 49

Figures

Figure 1-1 Example Structure Diagram 2
Figure 2-1 PVL Module Contents Syntax Diagram 5
Figure 2-2 White Space/Comment Syntax Diagram 5
Figure 2-3 Assignment Statement Syntax Diagram 6
Figure 2-4 Statement Delimiter Syntax Diagram 6
Figure 2-5 Value Syntax Diagram 7
Figure 2-6 Simple Value Syntax Diagram 8
Figure 2-7 String Type Syntax Diagram 10
Figure 2-8 Date Syntax Diagram 11
Figure 2-9 Time Syntax Diagram 12
Figure 2-10 Date/Time Syntax Diagram 12
Figure 2-11 Set Syntax Diagram 13
Figure 2-12 Sequence Syntax Diagram 13
Figure 2-13 Units Expression Syntax Diagram 14
Figure 2-14 Units Value Syntax Diagram 14
Figure 2-15 Aggregation Block Syntax Diagram 15
Figure 2-16 Aggregation Begin Statement Syntax Diagram 16
Figure 2-17 End Aggregation Statement Syntax Diagram 16
Figure 2-18 End Statement Syntax Diagram 18

Tables

Table 1-1 Reserved Character Set 3
Table 1-2 Unrestricted Character Set 4

REFERENCES

[1] "Procedures Manual for the Consultative Committee for Space Data Systems", CCSDS A00.0-Y-4, Yellow Book, Issue 4, Consultative Committee for Space Data Systems, September 1990.

[2] ISO 646-1991(E) Information Technology - ISO 7-bit coded character set for information interchange, Third Edition, 1991-12-15.

[3] ISO 8859-1-1987 Information Processing - 8-bit Single-byte coded graphic character set, Part 1: Latin Alphabet no.1.

[4] ISO 6093-1985(E) Information Processing - Representation of numerical values in character strings for information interchange, First Edition, 1985-11-01.

[5] ISO/IEC 8824:1990(E) Information Technology - Open System Interconnection - Specification of Abstract Syntax Notation One (ASN.1), Second Edition, 1990-12-15.

[6] Steedman, Douglas - Abstract Syntax Notation One (ASN.1) the Tutorial Reference, Technology Appraisals, London 1990.

[7] "Report Concerning Space Data Systems: Parameter Value Language -- A Tutorial", CCSDS 641.0-G-1, Consultative Committee for Space Data Systems, Green Book, Issue 1, May 1992 or later.

[8] "Recommendation for Space Data Systems: Time Code Formats", CCSDS 301.0-B-2, Consultative Committee for Space Data Systems, Blue Book, Issue 2, April 1990 or later.

This page intentionally left blank.

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 in order 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).

1.2 Applicability

The specifications in this document are to be invoked through the normal standards program of each member Agency 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 Annex A, which presents a complete summary of the acronyms and the terminology used in this document. Reference [7] 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 [7] prior to reading this document.

The document is structured as follows:

- Chapter 2 describes the PVL language, using English text and syntax diagrams.
- Chapter 3 provides the formal syntax specification written in Abstract Syntax Notation One (ASN.1, see Reference [5]). The comments in the ASN.1 are part of the specification. This is the ruling form of the specification.
- Annex A contains acronyms and a glossary of terms used in this document.
- Annex B lists the ASCII codes for the characters used in PVL.

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.

For example:

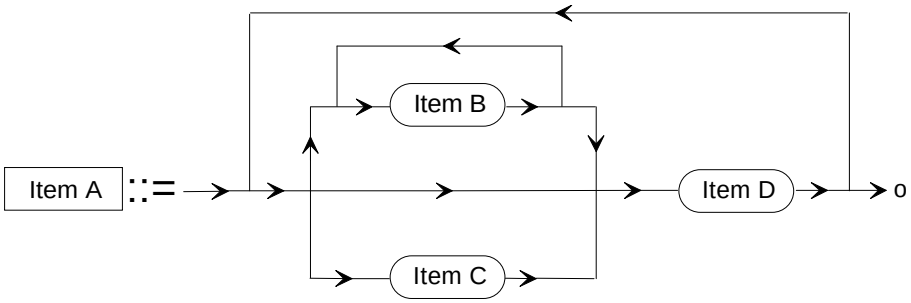


Figure 1-1 Example Structure Diagram

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) contain an Item A, the definition is recursive. Recursive structure definitions are permitted in this Recommendation.*

1.4 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.

1.4.1 PVL Character Set

The PVL Character Set is a subset of the ASCII character set. The specific subset is defined in Annex B. The PVL Character Set is split into three subsets: white space characters, reserved characters, and unrestricted characters.

1.4.1.1 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.

1.4.1.2 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 1-1.

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		

Table 1-1 Reserved Character Set

1.4.1.3 Unrestricted Character Set

The Unrestricted Character Set is a collection of PVL characters which are not reserved or used as white space. 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 1-2.

Symbol	Name	Symbol	Name	Symbol	Name
*	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)

Table 1-2 Unrestricted Character Set

1.4.2 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 asterisk-forward slash sequence (***/**). Comments are treated the same as white space when occurring between syntactic elements. Comments shall not be embedded within other comments.

2 OVERVIEW OF THE LANGUAGE

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 Section 2.3).

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.

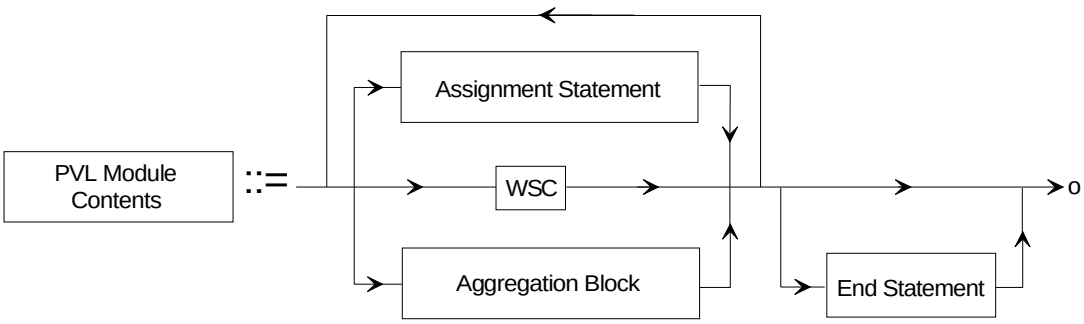


Figure 2-1 PVL Module Contents Syntax Diagram

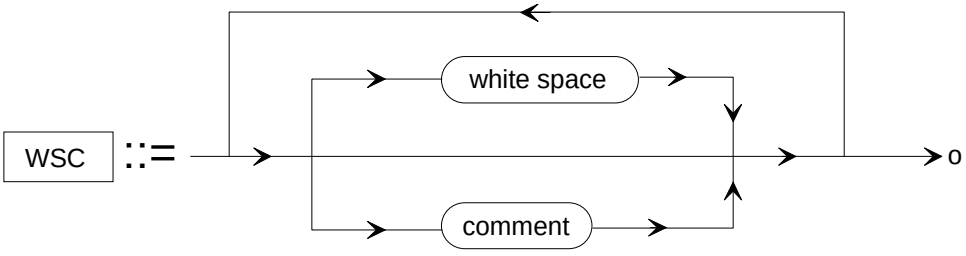


Figure 2-2 White Space/Comment Syntax Diagram

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

2.1 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.

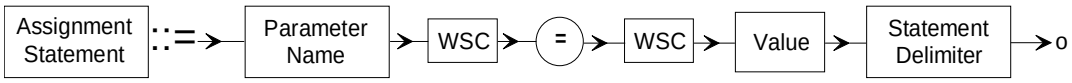


Figure 2-3 Assignment Statement Syntax Diagram

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

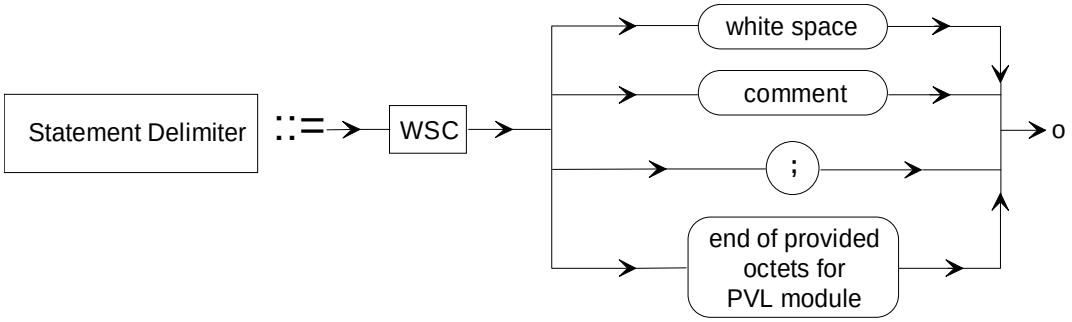


Figure 2-4 Statement Delimiter Syntax Diagram

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 set.

Statement delimitation in the absence of the explicit delimiter character is the only time when white space or comments have semantic meaning in PVL.

2.1.1 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 Section 2.1.2.1.1) or date/time encoding rules (see Section 2.1.2.1.3). A Parameter Name terminates with the character immediately prior to a Reserved Character, 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.

2.1.2 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.

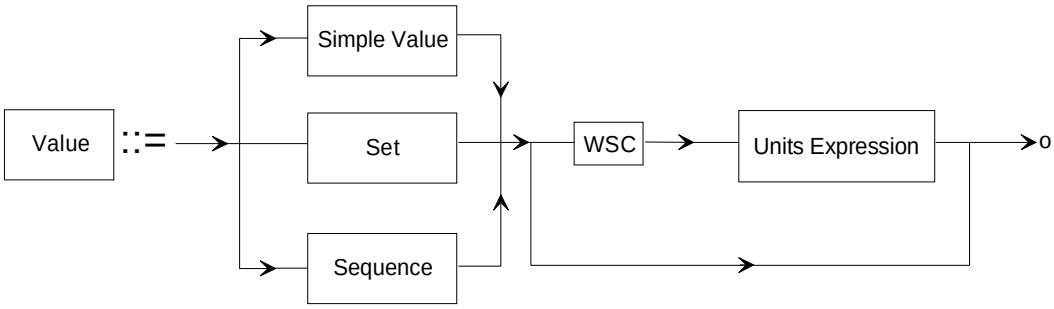


Figure 2-5 Value Syntax Diagram

2.1.2.1 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.

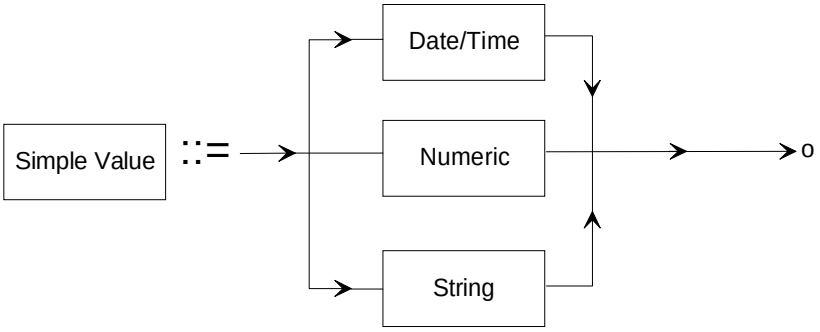


Figure 2-6 Simple Value Syntax Diagram

2.1.2.1.1 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.

2.1.2.1.1.1 Decimal Numbers

Decimal Numbers follow the three numerical representations (integer, floating point, exponential) specified in ISO 6093 (see Reference [4]) for decimal representations, with the exception that comma (**,**) shall not be used as a decimal point.

2.1.2.1.1.1.1 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
```

2.1.2.1.1.2 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.

2.1.2.1.1.1.3 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

2.1.2.1.1.2 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 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.*

2.1.2.1.1.2.1 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.*

2.1.2.1.1.2.2 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.*

2.1.2.1.1.2.3 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.*

2.1.2.1.2 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.

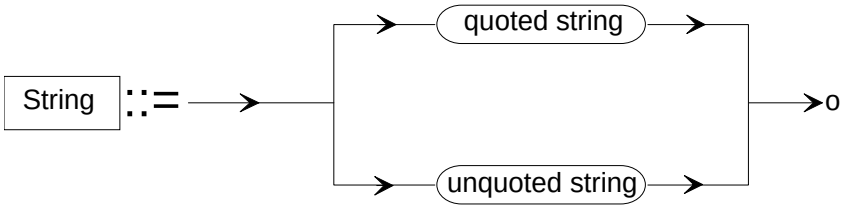


Figure 2-7 String Type Syntax Diagram

2.1.2.1.2.1 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 (**'**).

NOTE 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').*

NOTE 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.*

NOTE 3: *The above definition allows for null length (i.e., empty) strings. A null length string may have meaning and therefore is permitted.*

2.1.2.1.2.2 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 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.

2.1.2.1.3 Date/Time Value

The date/time value is a strict subset of the CCSDS ASCII Time Code recommendation (Reference [8]), 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.

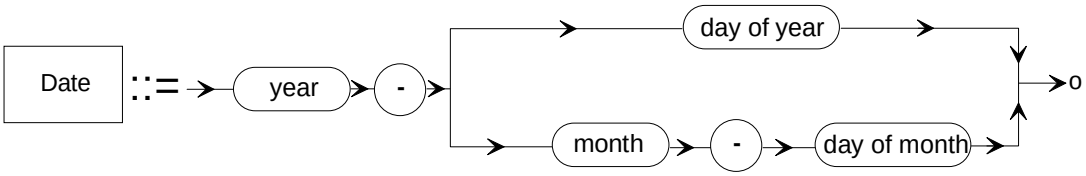


Figure 2-8 Date Syntax Diagram

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.

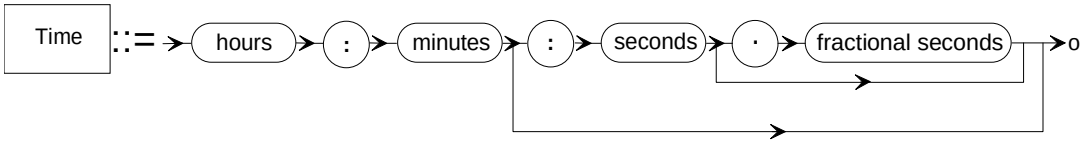


Figure 2-9 Time Syntax Diagram

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.

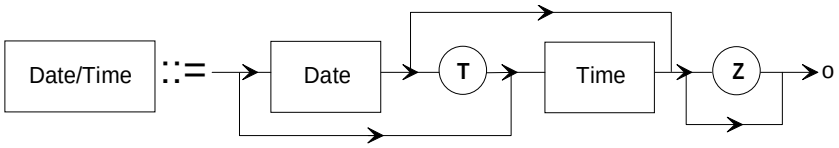


Figure 2-10 Date/Time Syntax Diagram

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

2.1.2.2 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 an 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.

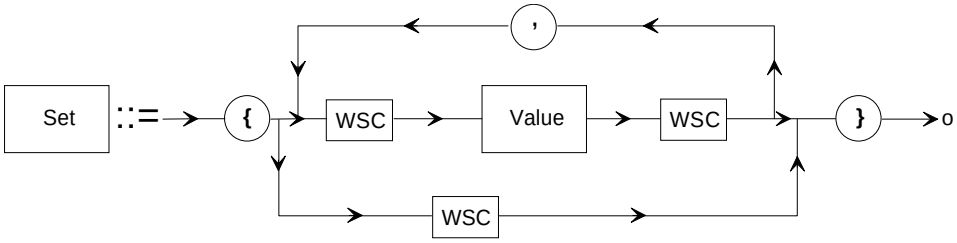


Figure 2-11 Set Syntax Diagram

2.1.2.3 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 an 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.

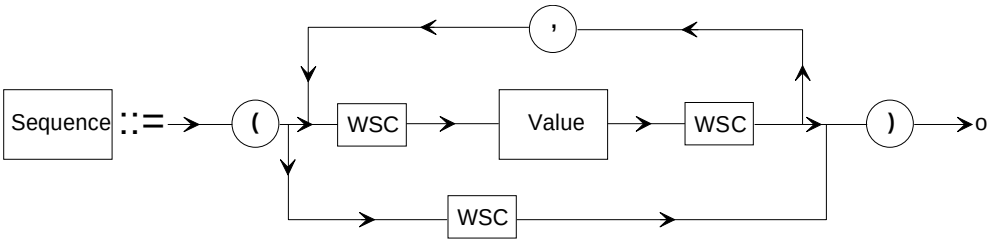


Figure 2-12 Sequence Syntax Diagram

2.1.2.4 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.

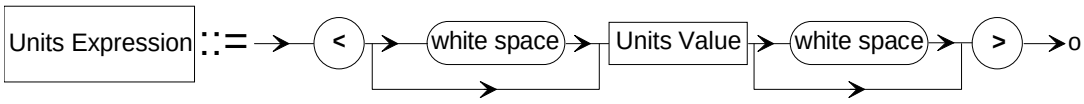


Figure 2-13 Units Expression Syntax Diagram

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.

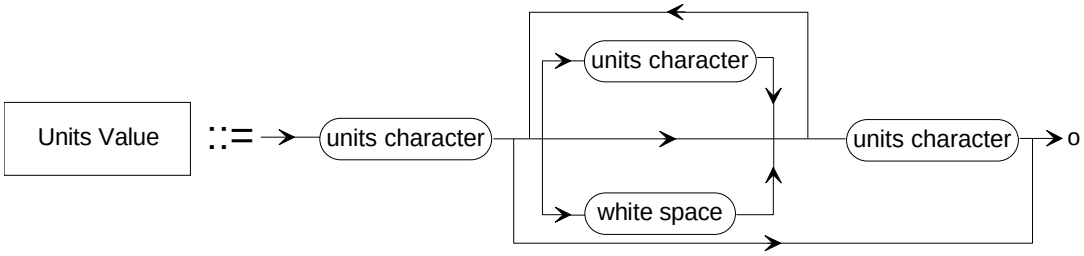


Figure 2-14 Units Value Syntax Diagram

2.2 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.

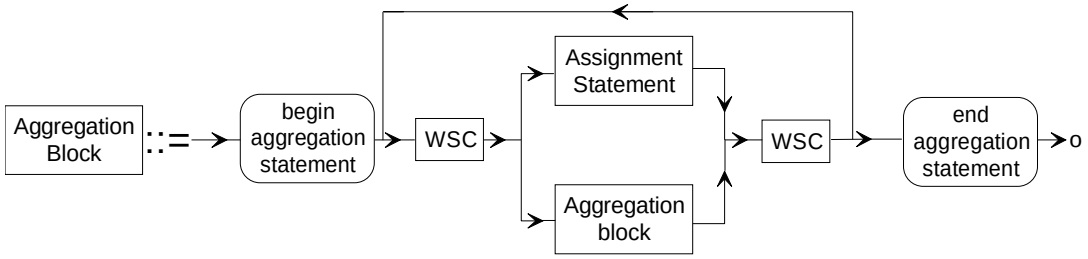


Figure 2-15 Aggregation Block Syntax Diagram

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 required.

2.2.1 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.

NOTE: *These synonyms are allowed for historical compatibility with several existing keyword languages.*

Figure 2-16 contains a syntax diagram of the begin aggregation statement formats.

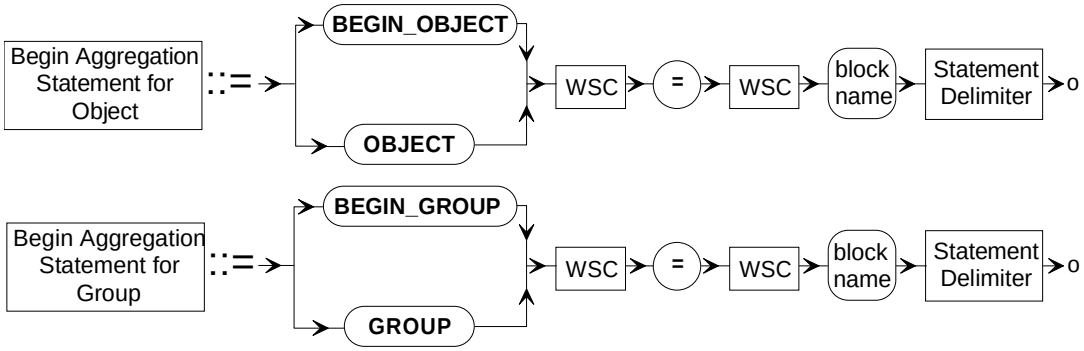


Figure 2-16 Aggregation Begin Statement Syntax Diagram

2.2.2 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 defined end-aggregation keywords are **END_GROUP** and **END_OBJECT**. Figure 2-17 contains a syntax diagram for the end aggregation statement.

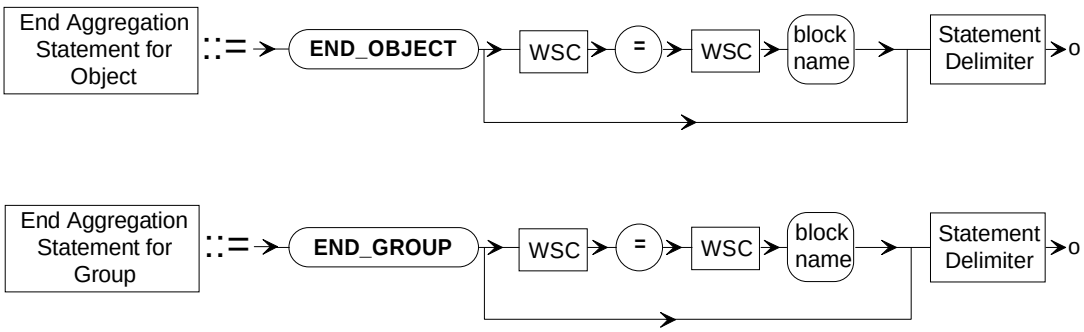


Figure 2-17 End Aggregation Statement Syntax Diagram

NOTE: *The preferred form of the aggregation end statement is the full form, which includes the block name.*

2.2.3 Aggregation Block Construction Rules

The end aggregation statement must be paired with an begin aggregation statement. In other words, an Aggregation Block which 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.

2.3 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

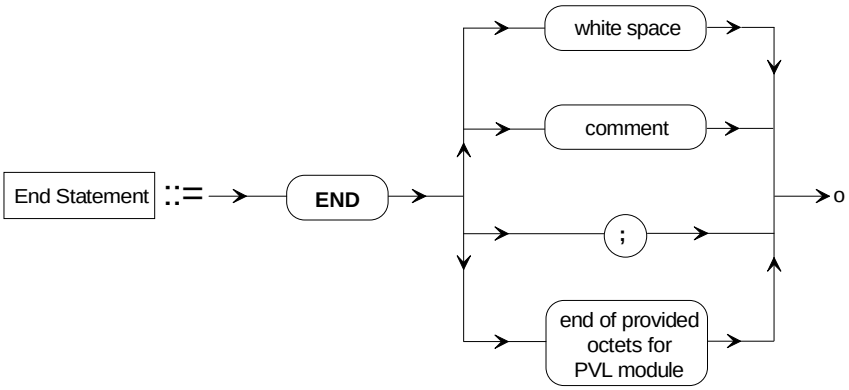


Figure 2-18 End Statement Syntax Diagram

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 semi-colons, 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 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[5]). 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 [6].

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:

PVL Module Contents 20
Aggregation Block 21
Assignment Statement 24
Comment 25
Date/Time 26
Numeric Values 32
Sequence 35
Set 36
String 37
Units Expression 39
Common Language Elements . . . 40

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

3.1 Formal Specification

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

```
                -- ******************************************************
                -- *****
                -- *****    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
                             SET 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
                            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
                            StatementDelim
                            }

BlockName               ::= SEQUENCE
                            {
                            -- Must not contain the sequence /* or */
                            -- Must not be reserved keyword (see Section 3.2)
                            -- 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
                        }

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

EndObjectLabel      ::= SEQUENCE
                        {
                        AssignmentSymbol,
                        WSC,
                        BlockName
                        -- Block Name must match Block Name
                        -- in paired Begin 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 Section 3.2)
                             -- 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 20
                            DecimalChar1to2,
                            PosDecimalChar
                            },
                        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 Section 3.2)
                                -- 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.


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

OpenAngleBracket        ::=  IA5String("<")

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

SemiColon               ::=  IA5String(";")

SeparatorSymbol         ::=  IA5String(",")
```

```
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,
                             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
```

3.2 Reserved Keywords

The following reserved keywords are not available for use as parameter names in assignment statements:

BEGIN_GROUP
BEGIN_OBJECT
END_GROUP
END_OBJECT
END
GROUP
OBJECT

ANNEX A -- ACRONYMS AND GLOSSARY

(THIS ANNEX **IS** A PART OF THE RECOMMENDATION)

Purpose:

This annex defines key acronyms and the glossary of terms which are used throughout this Recommendation to describe the concepts and elements of the Parameter Value Language

ANNEX A

Acronyms

ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
CCSDS	Consultative Committee for Space Data Systems
ISO	International Organization for Standardization
PVL	Parameter Value Language
SFDU	Standard Formatted Data Unit

Glossary of Terms

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 which terminates the PVL Module prior to the end of the provided octet space.

Numeric: A sequence of unrestricted characters that conform to encoding rules which 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 optionally terminated by a PVL end statement, within which PVL statements are written.

Reserved Characters: The set of PVL characters which may not occur in parameter names, unquoted strings, or block names.

Quote String Delimiters: The symbols apostrophe or quotation mark.

Quoted String: Zero or more PVL characters enclosed between matching quote string delimiters.

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

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.

Unquoted String: A value consisting of a sequence of unrestricted characters.

Unrestricted Characters: The set of PVL characters which may be used to form parameter names, unquoted strings or block names.

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

ANNEX B -- CHARACTER DEFINITIONS

(THIS ANNEX **IS** PART OF THIS SPECIFICATION.)

Purpose:

This annex contains the definition of the character representations used by the Parameter Value Language Specification.

PVL CHARACTER SET

The PVL Character set is a subset of the ASCII character set, specifically the 7-bit portion (MSB=0) of ISO 8859 G0 (Reference[3]) which set corresponds to ANSI 3.4-1977. This is also the same as the ISO 646 IRV set (Reference [2]), with the exception of positions 36 and 126, which display the currency symbol and overscore, respectively, in the IRV.

The PVL character set consists of the printable characters occupying the positions 33 to 126, inclusive (the Graphics Characters), the space (32), and the format effectors (positions 9 to 13 inclusive). These characters are listed on the following page.

Symbol	Name	Decimal Code	Symbol	Name	Decimal Code
	Horizontal tab	09	M	Capital letter M	77
	Line feed	10	N	Capital letter N	78
	Vertical tab	11	O	Capital letter O	79
	Form feed	12	P	Capital letter P	80
	Carriage return	13	Q	Capital letter Q	81
	Space	32	R	Capital letter R	82
!	Exclamation mark	33	S	Capital letter S	83
"	Quotation mark	34	T	Capital letter T	84
#	Number sign	35	U	Capital letter U	85
$	Dollar sign	36	V	Capital letter V	86
%	Percent sign	37	W	Capital letter W	87
&	Ampersand	38	X	Capital letter X	88
'	Apostrophe	39	Y	Capital letter Y	89
(	Left parenthesis	40	Z	Capital letter Z	90
)	Right parenthesis	41	[	Left square bracket	91
*	Asterisk	42	\	Reverse solidus	92
+	Plus sign	43	]	Right square bracket	93
,	Comma	44	^	Circumflex accent	94
-	Hyphen-minus	45	_	Low line, underline	95
.	Full stop	46	`	Grave accent	96
/	Solidus	47	a	Small letter a	97
0	Digit zero	48	b	Small letter b	98
1	Digit one	49	c	Small letter c	99
2	Digit two	50	d	Small letter d	100
3	Digit three	51	e	Small letter e	101
4	Digit four	52	f	Small letter f	102
5	Digit five	53	g	Small letter g	103
6	Digit six	54	h	Small letter h	104
7	Digit seven	55	i	Small letter i	105
8	Digit eight	56	j	Small letter j	106
9	Digit nine	57	k	Small letter k	107
:	Colon	58	l	Small letter l	108
;	Semicolon	59	m	Small letter m	109
<	Less than sign	60	n	Small letter n	110
=	Equals sign	61	o	Small letter o	111
>	Greater than sign	62	p	Small letter p	112
?	Question mark	63	q	Small letter q	113
@	Commercial at	64	r	Small letter r	114
A	Capital letter A	65	s	Small letter s	115
B	Capital letter B	66	t	Small letter t	116
C	Capital letter C	67	u	Small letter u	117
D	Capital letter D	68	v	Small letter v	118
E	Capital letter E	69	w	Small letter w	119
F	Capital letter F	70	x	Small letter x	120
G	Capital letter G	71	y	Small letter y	121
H	Capital letter H	72	z	Small letter z	122
I	Capital letter I	73	{	Left curly bracket	123
J	Capital letter J	74	|	Vertical line	124
K	Capital letter K	75	}	Right curly bracket	125
L	Capital letter L	76	~	Tilde	126

INDEX

Aggregation block 6, 15, 17, 19, 21, 45
Assignment statement 5-7, 15, 16, 19, 24, 45
Binary 8, 9, 32
Block name 15-17, 22, 23, 45
Comment 4, 5, 7, 10, 11, 18-20, 25, 41, 42, 45
Date/Time 7, 8, 10-12, 19, 22, 24, 26, 38
Decimal 8-10, 33, 48
End statement 5, 16, 18, 45
Explicit delimiter 6, 7
Exponential 8, 9, 32
Floating point 8
Formal specification 19
Hexadecimal 8-10, 34
Integer 8, 9, 32, 34
Numerics 8, 9
Octal 8, 9, 34
Parameter name 6, 7, 15, 45
PVL character set 3, 47
PVL module 5, 11, 18-20, 45
Quoted string 9, 10, 45
Reserved characters 2, 3, 10, 45
Sequence 13, 35, 45
Set 13, 36, 45
Simple value 7, 8, 14
Statement delimiter 6, 19, 41
String 8-11, 19, 24, 37, 45
Syntax diagram 5-8, 10-16, 18
Units expression 7, 14, 19, 39
Unquoted string 10, 11, 45
Unrestricted characters 3, 7, 8, 11, 45
Value 5-8, 13, 24, 35, 36, 45
White space 3-7, 10, 11, 14, 18, 19, 45