
British Standard Specification for

Programming language: APL

Languages de programmation — APL

Programmiersprachen; APL

National foreword

This British Standard has been prepared under the direction of the Information Systems Technology Standards Policy Committee. It is identical with ISO 8485 : 1989 'Programming languages — APL' published by the International Organization for Standardization (ISO).

The UK actively participated through BSI Technical Committee IST/5 in the development of the ISO standard.

Cross-references. The Technical Committee has reviewed the provisions of ISO 2375 and ISO 2382-15 to which reference is made in the text, and has decided that they are acceptable for use in conjunction with this standard.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Amendment No. 0
published and effective from 15 February 1994
to BS 7301 : 1990

Specification for programming language: APL

NOTE. The European Committee for Standardization (CEN) has accepted ISO 8485 : 1989 as a European Standard designated as EN 28485 : 1993. This amendment implements EN 28485 : 1993 as a British Standard in the BS EN series.

Implementation of European Standard

Front cover

Delete the existing cover page and substitute the attached.

AMD 8034/February 1994

National foreword

At the end of paragraph 1 insert the following new paragraph.

'In 1993 the European Committee for Standardization (CEN) accepted ISO 8485 : 1989 as European Standard EN 28485. As a consequence of implementing the European Standard this British Standard is renumbered as BS EN 28485 and any reference to BS 7301 should be read as a reference to BS EN 28485.'

AMD 8034/February 1994

New EN title page and foreword

Immediately after the national foreword, insert the attached new EN title page and foreword page.

AMD 8034/February 1994

Programming language: APL

The European Standard EN 28485 : 1993 has the status of a
British Standard

Langages de programmation — APL

Programmiersprachen — APL

UDC 681.3.04 : 800.92

UDC 681.3.04 : 800.92

Descriptors: Data processing, programming: computers, programming languages, APL

English version

Programming languages — APL

(ISO 8485 : 1989)

Langages de programmation — APL
(ISO 8485 : 1989)

Programmierspachen — APL
(ISO 8485 : 1989)

This European Standard was approved by CEN on 1993-08-13. CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the Central Secretariat has the same status as the official versions.

CEN members are the national standards bodies of Austria, Belgium, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and United Kingdom.

CEN

European Committee for Standardization
Comité Européen de Normalisation
Europäisches Komitee für Normung

Central Secretariat: rue de Stassart 36, B-1050 Brussels

Foreword

On the proposal of the CEN Central Secretariat, the Technical Board has decided to submit the International Standard:
'Programming languages — APL (ISO 8485 : 1989)' to the formal vote.

The result of the formal vote was positive.

For the time being, this document exists only in English and in French.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by February 1994 and conflicting national standards shall be withdrawn at the latest by February 1994.

In accordance with the CEN/CENELEC Internal Regulations, the following countries are bound to implement this European Standard:

Austria, Belgium, Denmark, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland and United Kingdom.

Contents

	Page
National foreword	Inside front cover
Committees responsible	Back cover

Specification

0 Introduction	3
1 Scope and Field of Application	5
2 References	7
3 Form of the Standard	9
3.1 Form of Definitions	9
3.2 Named Arrays in Examples	10
3.3 Notes	10
3.4 Cross-References	10
3.5 General Definitions	11
4 Compliance	13
4.1 Conforming Implementations	13
4.1.1 Required Behaviour for Conforming Implementations	13
4.1.2 Required Documentation for Conforming Implementations	14
4.1.2.1 Documentation of Optional-Facilities	14
4.1.2.2 Documentation of Implementation-Defined-Facilities	14
4.1.2.3 Consistent Extensions	14
4.2 Conforming Programs	15
4.2.1 Required Behaviour for Conforming Programs	15
4.2.2 Required Documentation for Conforming Programs	15
5 Definitions	17
5.1 Characters	17
5.2 Numbers	19
5.2.1 Elementary Operations	19
5.2.2 Number Constants	20
5.2.3 Subsets of the Set of Numbers	21
5.2.4 Implementation Algorithms	23
5.2.5 Defined Operations	24
5.3 Objects	26
5.3.1 Lists	26
5.3.2 Arrays	27
5.3.3 Defined-Functions	29
5.3.4 Tokens	30
5.3.4.1 Metaclasses	32
5.3.4.2 Index-List	33
5.3.5 Symbols	33
5.3.6 Contexts	34
5.3.7 Workspaces	34
5.3.8 Sessions	35
5.3.9 Shared-Variables	36
5.3.10 Systems	37
5.4 Evaluation Sequences	39
5.4.1 Evaluation Sequence Phrases	39
5.4.2 Diagrams	41
5.5 Other Terms	42

6 Syntax and Evaluation		43
6.1 Introduction		43
6.1.1 Evaluate-Line		43
6.1.2 Character-Diagrams		45
6.1.3 Evaluate-Statement		52
6.1.4 Bind-Token-Class		53
6.1.5 Literal-Conversion		54
6.1.6 Statement-Analysis Token-Diagrams		55
6.2 Reduce-Statement		60
6.3 The Phrase Evaluators		63
6.3.1 Diagrams		64
6.3.2 Remove-Parentheses		65
6.3.3 Evaluate-Niladic-Function		65
6.3.4 Evaluate-Monadic-Function		66
6.3.5 Evaluate-Monadic-Operator		68
6.3.6 Evaluate-Dyadic-Function		69
6.3.7 Evaluate-Dyadic-Operator		71
6.3.8 Evaluate-Indexed-Reference		72
6.3.9 Evaluate-Assignment		73
6.3.10 Evaluate-Indexed-Assignment		74
6.3.11 Evaluate-Variable		75
6.3.12 Build-Index-List		76
6.3.13 Process-End-of-Statement		77
6.4 The Form Table		78
7 Scalar Functions		83
7.1 Monadic Scalar Functions		84
7.1.1 Conjugate	$Z \leftarrow + B$	84
7.1.2 Negative	$Z \leftarrow - B$	84
7.1.3 Signum	$Z \leftarrow \times B$	85
7.1.4 Reciprocal	$Z \leftarrow \div B$	85
7.1.5 Floor	$Z \leftarrow \lfloor B$	86
7.1.6 Ceiling	$Z \leftarrow \lceil B$	86
7.1.7 Exponential	$Z \leftarrow * B$	87
7.1.8 Natural Logarithm	$Z \leftarrow \otimes B$	87
7.1.9 Magnitude	$Z \leftarrow B$	88
7.1.10 Factorial	$Z \leftarrow ! B$	88
7.1.11 Pi times	$Z \leftarrow \circ B$	89
7.1.12 Not	$Z \leftarrow \sim B$	89
7.2 Dyadic Scalar Functions		90
7.2.1 Plus	$Z \leftarrow A + B$	91
7.2.2 Minus	$Z \leftarrow A - B$	91
7.2.3 Times	$Z \leftarrow A \times B$	92
7.2.4 Divide	$Z \leftarrow A \div B$	92
7.2.5 Maximum	$Z \leftarrow A \lceil B$	93
7.2.6 Minimum	$Z \leftarrow A \lfloor B$	93
7.2.7 Power	$Z \leftarrow A * B$	94
7.2.8 Logarithm	$Z \leftarrow A \otimes B$	94
7.2.9 Residue	$Z \leftarrow A B$	95
7.2.10 Binomial	$Z \leftarrow A ! B$	96
7.2.11 Circular Functions	$Z \leftarrow A \circ B$	97
7.2.12 And	$Z \leftarrow A \wedge B$	99
7.2.13 Or	$Z \leftarrow A \vee B$	99
7.2.14 Nand	$Z \leftarrow A \nabla B$	100
7.2.15 Nor	$Z \leftarrow A \nabla B$	100
7.2.16 Equal	$Z \leftarrow A = B$	101
7.2.17 Less than	$Z \leftarrow A < B$	103
7.2.18 Less than or equal to	$Z \leftarrow A \leq B$	103
7.2.19 Not equal	$Z \leftarrow A \neq B$	104
7.2.20 Greater than or equal to	$Z \leftarrow A \geq B$	104
7.2.21 Greater than	$Z \leftarrow A > B$	105

8	Structural Primitive Functions		107
8.1	Introduction		107
8.2	Monadic Structural Primitive Functions		107
8.2.1	Ravel	$Z \leftarrow , B$	107
8.2.2	Shape	$Z \leftarrow \rho B$	108
8.2.3	Index Generator	$Z \leftarrow \iota B$	109
8.3	Dyadic Structural Primitive Functions		110
8.3.1	Reshape	$Z \leftarrow A \rho B$	110
8.3.2	Join	$Z \leftarrow A , B$	111
9	Operators		113
9.1	Introduction		113
9.2	Monadic Operators		114
9.2.1	Reduction	$Z \leftarrow f/ B$	114
9.2.1	Reduction	$Z \leftarrow f/[K] B$	114
9.2.1	Reduction	$Z \leftarrow f\overline{/} B$	114
9.2.1	Reduction	$Z \leftarrow f\overline{/}[K] B$	114
9.2.2	Scan	$Z \leftarrow f\backslash B$	116
9.2.2	Scan	$Z \leftarrow f\backslash[K] B$	116
9.2.2	Scan	$Z \leftarrow f\backslash B$	116
9.2.2	Scan	$Z \leftarrow f\backslash[K] B$	116
9.3	Dyadic Operators		117
9.3.1	Outer Product	$Z \leftarrow A \circ . f B$	117
9.3.2	Inner Product	$Z \leftarrow A f . g B$	118
10	Mixed Functions		119
10.1	Monadic Mixed Functions		119
10.1.1	Roll	$Z \leftarrow ? B$	119
10.1.2	Grade Up	$Z \leftarrow \Delta B$	121
10.1.3	Grade Down	$Z \leftarrow \nabla B$	122
10.1.4	Reverse	$Z \leftarrow \phi B$	123
10.1.4	Reverse	$Z \leftarrow \theta B$	123
10.1.4	Reverse	$Z \leftarrow \phi[K] B$	123
10.1.4	Reverse	$Z \leftarrow \theta[K] B$	123
10.1.5	Monadic Transpose	$Z \leftarrow \mathfrak{q} B$	124
10.1.6	Matrix Inverse	$Z \leftarrow \mathfrak{g} B$	125
10.1.7	Execute	$Z \leftarrow \mathfrak{e} B$	126
10.2	Dyadic Mixed Functions		127
10.2.1	Join Along an Axis	$Z \leftarrow A , [K] B$	127
10.2.2	Index of	$Z \leftarrow A \iota B$	130
10.2.3	Member of	$Z \leftarrow A \in B$	131
10.2.4	Deal	$Z \leftarrow A ? B$	132
10.2.5	Compress	$Z \leftarrow A / B$	133
10.2.5	Compress	$Z \leftarrow A \overline{/} B$	133
10.2.5	Compress	$Z \leftarrow A / [K] B$	133
10.2.5	Compress	$Z \leftarrow A \overline{/} [K] B$	133
10.2.6	Expand	$Z \leftarrow A \backslash B$	135
10.2.6	Expand	$Z \leftarrow A \backslash B$	135
10.2.6	Expand	$Z \leftarrow A \backslash [K] B$	135
10.2.6	Expand	$Z \leftarrow A \backslash [K] B$	135
10.2.7	Rotate	$Z \leftarrow A \phi B$	137
10.2.7	Rotate	$Z \leftarrow A \theta B$	137
10.2.7	Rotate	$Z \leftarrow A \phi [K] B$	137
10.2.7	Rotate	$Z \leftarrow A \theta [K] B$	137
10.2.8	Base Value	$Z \leftarrow A \perp B$	139
10.2.9	Representation	$Z \leftarrow A \tau B$	140
10.2.10	Dyadic Transpose	$Z \leftarrow A \mathfrak{q} B$	142
10.2.11	Take	$Z \leftarrow A \uparrow B$	144
10.2.12	Drop	$Z \leftarrow A \downarrow B$	145
10.2.13	Matrix Divide	$Z \leftarrow A \mathfrak{g} B$	146
10.2.14	Indexed Reference	$Z \leftarrow A[I]$	147
10.2.15	Indexed Assignment	$Z \leftarrow V[I] \leftarrow B$	148

11 System Functions		151
11.1 Introduction		151
11.2 Definition		151
11.3 Diagram		151
11.4 Niladic System Functions		152
11.4.1 Time Stamp	$Z \leftarrow \square TS$	152
11.4.2 Atomic Vector	$Z \leftarrow \square AV$	153
11.4.3 Line Counter	$Z \leftarrow \square LC$	153
11.5 Monadic System Functions		154
11.5.1 Delay	$Z \leftarrow \square DL B$	154
11.5.2 Name Class	$Z \leftarrow \square NC B$	155
11.5.3 Expunge	$Z \leftarrow \square EX B$	156
11.5.4 Name List	$Z \leftarrow \square NL B$	156
11.5.5 Query Stop	$Z \leftarrow \square STOP B$	157
11.5.6 Query Trace	$Z \leftarrow \square TRACE B$	157
11.6 Dyadic System Functions		158
11.6.1 Name List	$Z \leftarrow A \square NL B$	158
11.6.2 Set Stop	$Z \leftarrow A \square STOP B$	158
11.6.3 Set Trace	$Z \leftarrow A \square TRACE B$	159
12 System Variables		161
12.1 Definitions		161
12.2 Evaluation Sequences		162
12.2.1 Comparison Tolerance	$Z \leftarrow \square CT \leftarrow B$	162
12.2.1 Comparison Tolerance	$Z \leftarrow \square CT$	162
12.2.2 Random Link	$Z \leftarrow \square RL \leftarrow B$	163
12.2.2 Random Link	$Z \leftarrow \square RL$	163
12.2.3 Print Precision	$Z \leftarrow \square PP \leftarrow B$	164
12.2.3 Print Precision	$Z \leftarrow \square PP$	164
12.2.4 Index Origin	$Z \leftarrow \square IO \leftarrow B$	165
12.2.4 Index Origin	$Z \leftarrow \square IO$	165
12.2.5 Latent Expression	$Z \leftarrow \square LX \leftarrow B$	166
12.2.5 Latent Expression	$Z \leftarrow \square LX$	166
12.2.5 Latent Expression	$Z \leftarrow \square LX[I] \leftarrow B$	166
13 Defined Functions		167
13.1 Introduction		167
13.2 Definitions		168
13.3 Diagrams		171
13.4 Operations		174
13.4.1 Call-Defined-Function	$Z \leftarrow DFN$	174
13.4.1 Call-Defined-Function	$Z \leftarrow DFN B$	174
13.4.1 Call-Defined-Function	$Z \leftarrow A DFN B$	174
13.4.2 Defined-Function-Control		175
13.4.3 Function Fix	$Z \leftarrow \square FX B$	176
13.4.4 Character Representation	$Z \leftarrow \square CR B$	177
13.5 Function Editing		178
13.5.1 Evaluate-Function-Definition-Request		178
13.5.2 Evaluate-Editing-Request		180
13.5.3 Diagrams		181
14 Shared Variables		183
14.1 Informal Introduction		183
14.2 Definitions		186
14.3 Diagrams		186
14.4 Operations		186
14.4.1 Primary-Name		186
14.4.2 Surrogate-Name		186
14.4.3 Degree-of-Coupling		186
14.4.4 Access-Control-Vector		186
14.4.5 Offer		187
14.4.6 Retract		188
14.4.7 Shared-Variable-Reset		188

14.5 Shared Variable Forms		189
14.5.1 Shared Variable Reference	$Z \leftarrow SHV$	189
14.5.2 Shared Variable Assignment	$Z \leftarrow SHV \leftarrow B$	189
14.5.3 Shared Variable Indexed Assignment	$Z \leftarrow SHV[I] \leftarrow B$	190
14.6 Shared Variable System Functions		191
14.6.1 Shared Variable Access Control Inquiry	$Z \leftarrow \square SVC \ B$	191
14.6.2 Shared Variable Query	$Z \leftarrow \square SVQ \ B$	192
14.6.3 Shared Variable Degree of Coupling	$Z \leftarrow \square SVO \ B$	193
14.6.4 Shared Variable Offer	$Z \leftarrow A \ \square SVO \ B$	194
14.6.5 Shared Variable Retraction	$Z \leftarrow \square SVR \ B$	195
14.6.6 Shared Variable Access Control Set	$Z \leftarrow A \ \square SVC \ B$	196
15 Formatting and Numeric Conversion		197
15.1 Introduction		197
15.2 Numeric Conversion		197
15.2.1 Numeric-Input-Conversion		198
15.2.2 Numeric-Output-Conversion		199
15.3 Diagrams		199
15.4 Operations		203
15.4.1 Monadic Format	$Z \leftarrow \Phi \ B$	203
15.4.2 Dyadic Format	$Z \leftarrow A \ \Phi \ B$	206
16 Input and Output		209
16.1 Introduction		209
16.2 Definitions		209
16.2.1 User Facilities		209
16.2.2 Implementation Algorithms		210
16.2.3 Prompts		211
16.3 Diagrams		211
16.4 Operations		212
16.4.1 Immediate-Execution		212
16.4.2 Quad Input	$Z \leftarrow \square$	213
16.4.3 Quote Quad Input	$Z \leftarrow \square$	214
16.4.4 Quad Output	$Z \leftarrow \square \leftarrow B$	214
16.4.5 Quote Quad Output	$Z \leftarrow \square \leftarrow B$	215
17 System Commands		217
17.1 Introduction		217
17.2 Definitions		217
17.3 Diagrams		218
17.4 Operations		219
17.4.1 Evaluate-System-Command		219
17.5 Diagrams and Evaluation Sequences		220
Annex A. The APL Character Set for Workspace Interchange		225
Annex B: The Workspace Interchange Convention		233
B.1 Introduction		233
B.1.1 Purpose of the Convention		233
B.1.2 Organisation of the Convention		233
B.2 Level 1 Description		234
B.2.1 Canonical Representation Vectors		234
B.2.2 Representation of Variables		235
B.2.3 Representation of Functions		236
B.2.4 Representation of Executable Expressions		237
B.3 Level 2 Description		238
B.3.1 Multiple Canonical Representation Vectors		238
B.3.2 Workspace Environment		238
B.3.3 Pseudovariables		239
B.3.4 Interchange Convention Identifier		239
B.3.5 Escape		239
B.3.6 Bits		239
B.3.7 Translate		239
B.3.8 Note		240
B.3.9 Workspace Identifier		240
B.3.10 End of the Stream		240

B.3.11 Inclusion of Nonstandard Material 240
 B.3.12 Use of Other Types 241
 B.4 Level 3 Description 242
 B.4.1 Atoms 242
 B.4.2 Registered character sets 242
 B.4.3 Escape sequences 242
 B.4.4 Registered characters 242
 B.4.5 ISO-APL characters 242
 B.4.6 Characters 243
 B.4.7 The Translate Table 243
 B.4.8 Example 244
 B.4.9 Escape 244
 B.4.10 Bits 244
 B.4.11 Translate 244
 B.4.12 Example 245
 B.4.13 Mapping of Characters to Bits 245
 B.4.14 Stage 1 Translation 245
 B.4.15 Stage 2 Translation 245
 B.4.16 Example 246
 B.4.17 Summary 246
 B.5 Level 4 Description 247
 B.5.1 Physical Media 247
 B.5.2 Nine-track Magnetic Tape 247
 B.6 Extended Example 248

 Bibliography 253

 INDEX 255

Programming languages — APL

0 INTRODUCTION

APL stands for **A Programming Language**. It is a notation invented by K. E. Iverson in the late 1950s for the description of algorithms, and expanded on and made into the programming system *APL\360* by Iverson and his colleagues Adin Falkoff, Larry Breed, Dick Lathwell, and Roger Moore in the mid-1960s.

Throughout this document

- the term “this standard” is understood to mean “this International Standard”;
- the words “chapter”, “section” and “subsection” are understood to mean “clause”, “subclause” and “sub-subclause”, respectively.

1 SCOPE AND FIELD OF APPLICATION

This standard defines the programming language APL and the environment in which APL programs are executed. Its purpose is to facilitate interchange and promote portability of APL programs and programming skills.

This standard specifies the syntax and semantics of APL programs and the characteristics of the environment in which APL programs are executed.

It also specifies requirements for conformance to this standard, including the publication of values and characteristics of implementation properties so that conforming implementations can be meaningfully compared.

This standard does not specify:

- implementation properties that are likely to vary with the particular equipment or operating system used;
- required values for implementation limits such as APL workspace size or numeric precision;
- the data structures used to represent APL objects;
- the facilities available through shared variables.

2 REFERENCES

ISO 2375 : 1985, *Data processing — Procedure for registration of escape sequences.*

ISO 2382-15 : 1985, *Data processing — Vocabulary — Part 15: Programming languages.*

3 FORM OF THE STANDARD

This standard is a formal model of an APL machine, specified as a collection of finite sets, diagrams, and evaluation sequences, and objects constructed from finite sets, diagrams, and evaluation sequences.

The finite sets are the **implementation-defined character-set**, the **implementation-defined set of numbers**, and the enumerated sets **array-type**, **class-names**, **keyboard-states**, **mode-names**, **required-character-set**, and **workspace-presence**.

Diagrams are directed graphs used to designate syntactic forms.

Evaluation sequences are formal procedures that operate on finite sets, diagrams, other evaluation sequences and objects defined in the standard.

Objects are entities consisting of enumerated set members and other objects. The objects are **list**, **array**, **defined-function**, **token**, **symbol**, **context**, **workspace**, **session**, **shared-variable**, and **system**.

Each object has attributes describing its state. The attributes of an array, for example, are its typical element, its shape, and its ravel.

Objects often have defined properties derived from their attributes. The rank of an array, for example, is the shape of the shape of the array.

3.1 FORM OF DEFINITIONS

Defined terms in this standard are always set in **bold** and indexed. The index entry begins with the page number of the definition followed by the page numbers of all references to the term. If the definition and a use of a term occur on the same page, that page number will occur twice in the Index.

The following terms occur throughout the document and are not cross-indexed: **character**, **content**, **class**, **item**, and **number**.

*Note: Terms in this document include both phrases such as **implementation-parameter** and words such as **nil**.*

Each definition in this standard takes one of four forms.

- (1) Regular definitions consist of the term being defined followed by a colon and the body of the definition. The term **Boolean** is defined in this way.
- (2) Members of an enumerated set are defined simply by being listed in the definition of the enumerated set. The term **nil** is defined in this way, as a member of the enumerated set **class-names**.
- (3) Diagrams are defined by directed graphs. The term **expression** is defined in this way.
- (4) Definitions of terms that designate evaluation sequences take the following form:

The term being defined, such as **scan**.

The forms that the evaluation mechanism used in the standard recognises as designating this term, such as $Z \leftarrow f \backslash B$.

An informal introduction indicating the purpose of the procedure. The informal introduction is considered commentary on the standard.

An evaluation sequence expressed in a formal, though English-like, language defined in the subsection **Evaluation Sequences**. A **conforming-implementation** is required to emulate the behaviour described in the evaluation sequence as modified by the **additional-requirements**, if any.

Examples, which show effects of the procedure specified by the evaluation sequence. Examples are considered commentary on the standard.

Additional Requirements, giving aspects of the behaviour required of this operation that cannot conveniently be expressed in the evaluation sequence.

3.2 NAMED ARRAYS IN EXAMPLES

In the examples in this standard, APL identifiers beginning with *N*, such as *N234*, represent numeric arrays whose shape and content are specified by the digits in the identifier. Each digit in the identifier specifies an element of the shape vector; each element of the array, when broken down into digits, gives the index of that element.

For example,

```

      N4
1 2 3 4
      N23
11 12 13
21 22 23
      N234
111 112 113 114
121 122 123 124
131 132 133 134

211 212 213 214
221 222 223 224
231 232 233 234
      N234[2;3;1]
231

```

3.3 NOTES

This standard contains notes that comment on the text of the standard, pointing out the significance of definitions, noting relationships between definitions, and otherwise making the text approachable. These notes are set in a different type style than the text of the standard, and are prefixed with the word "Note:". The following is an example of a note.

Note: This is an example of a note.

Notes never set requirements for conformance. They may suggest desired properties, but such suggestions are not mandatory for conformance.

3.4 CROSS-REFERENCES

The heading levels in this standard are chapter, section, and subsection. When cross-references are given, they are always to a subsection title. In the Index, subsections are treated like definitions: the page on which a subsection begins is always the first entry in the Index; subsequent page numbers in the index show where references are made to the subsection.

3.5 GENERAL DEFINITIONS

For the purpose of this standard, the definitions given in ISO 2382-15 and the following definitions apply:

3.5.1 **Program:** An application.

Note: The term is used in this standard to include everything from an APL expression to a collection of workspaces communicating via shared variables.

3.5.2 **Implementation:** A combination of computer hardware and software that processes (APL) programs.

*Note: An implementation is an instance of the object **system** specified by this standard.*

3.5.3 **Facility** (of an implementation): A unit of behaviour. Every **facility** is one of:

Defined-Facility: A **facility** fully specified in this standard and not designated **optional** or **implementation-defined**.

Optional-Facility: A **facility** fully specified in this standard and designated **optional**.

Implementation-Defined-Facility: A **facility** not fully specified by this standard that is designated **implementation-defined**.

Consistent-Extension: A **facility** not specified in this standard that, for a construct this standard specifies as producing an error, gives some effect other than signalling the specified error.

4 COMPLIANCE

4.1 CONFORMING IMPLEMENTATIONS

An APL **implementation** conforms to this standard if it meets the following requirements for both behaviour and documentation.

4.1.1 Required Behaviour for Conforming Implementations

A **conforming-implementation** shall provide all **defined-facilities** and **implementation-defined-facilities**. Each such **facility** shall behave as specified by this standard.

A **conforming-implementation** may provide **optional-facilities**. If provided, an **optional-facility** shall behave as specified by this standard. Attempted use of an **optional-facility** that is not provided shall cause the **conforming-implementation** to signal an error. A **conforming-implementation** shall not replace the error signalled by a missing **optional-facility** with other behaviour.

A **conforming-implementation** may provide **consistent-extensions**. The presence of a **consistent-extension** shall not affect the behaviour of a **conforming-program**.

A **conforming-implementation** shall use algorithms that produce results that are the same as those produced by the evaluation sequences. Mathematical function algorithms shall have at least the accuracy that the algorithms given in the evaluation sequences would produce.

Note: The evaluation sequences used in this standard are intended to specify results, not implementation techniques.

*The errors produced by the absence of an **optional-facility** cannot be replaced by **consistent-extensions** in a **conforming-implementation**, since this would affect the behaviour of **conforming-programs** that use the **optional-facility**.*

4.1.2 Required Documentation for Conforming Implementations

A **conforming-implementation** shall provide a reference document that satisfies the following requirements for the documentation of **optional-facilities**, **implementation-defined-facilities**, and **consistent-extensions**:

4.1.2.1 Documentation of Optional-Facilities

A **conforming-implementation** shall document the presence or absence of each of the following **optional-facilities**:

Shared-Variable-Protocol. A mechanism that permits one session to exchange data with other autonomous sessions.

Statement-Separator-Facility. A mechanism for placing more than one statement on a line.

Trace-and-Stop-Control. A mechanism that assists the testing and correction of defined functions.

4.1.2.2 Documentation of Implementation-Defined-Facilities

A **conforming-implementation** shall document the following aspects of **implementation-defined-facilities**:

A description of the **character-set**. This shall include a table showing the correspondence between index positions in the **atomic-vector** and the members of the **required-character-set**. If the graphic symbols used in a **conforming-implementation** are dissimilar to those in **Table 1**, the correspondence between the graphic symbols used in the implementation and those in **Table 1** shall be given.

A description of the **numbers**. This shall include a characterisation of the internal representation used for **numbers**.

Descriptions of the characteristics of each **implementation-algorithm**.

The value of each **implementation-parameter**.

A description of each **internal-value-set**.

4.1.2.3 Consistent Extensions

A **conforming-implementation** shall document all **consistent-extensions** it provides. The documentation shall clearly indicate that the use of a **consistent-extension** prevents a **program** from conforming with this standard.

*Note: Implementers of **conforming-implementations** should, in general, be wary of replacing **limit-errors** with **consistent-extensions**, since these errors are the only safeguards a **conforming-program** has when attempting to operate in a **conforming-implementation** whose **implementation-parameters** are inadequate to support it.*

*For example, if the **limit-error** on **identifier-length-limit** were not signalled, a **conforming-program** with identifiers longer than the local **identifier-length-limit** would malfunction without warning.*

4.2 CONFORMING PROGRAMS

A **program** conforms to this standard if it meets the following requirements for both behaviour and documentation.

4.2.1 Required Behaviour for Conforming Programs

A **conforming-program** shall use only those **facilities** specified in this standard.

A **conforming-program** shall not use **consistent-extensions**.

A **conforming-program** shall not depend on the signalling of any error by a **conforming-implementation**.

*Note: Conforming-programs cannot depend on error behaviour, since **consistent-extensions** that replace errors are permitted in conforming-implementations.*

4.2.2 Required Documentation for Conforming Programs

A **conforming-program** shall document which **optional-facilities** it requires.

A **conforming-program** shall document any specific requirements it has for **implementation-parameters**.

*Note: A conforming-program may or may not work, and may or may not produce identical results on all conforming-implementations, because of inherent dependencies on **implementation-parameters** or **implementation-algorithms** such as the algorithm used for matrix inversion.*

*It is suggested that **conforming-programs** provide documentation detailing the values of **implementation-parameters** they require so that their suitability for a given **conforming-implementation** can be determined readily.*

5 DEFINITIONS

5.1 CHARACTERS

Character-Set: An **implementation-defined** finite set.

Character: A member of the **implementation-defined** finite set **character-set**.

Required-Character-Set: An enumerated set whose members are designated by the graphic symbols in **Table 1**. The **character-set** provided by a **conforming-implementation** shall contain all members of the **required-character-set**.

Additional Requirement:

Members of the **required-character-set** are represented in this standard by graphic symbols in a particular typeface, as shown in **Table 1**. The graphic symbols associated with the **required-character-set** in a **conforming-implementation** are **implementation-defined**.

*Note: There are conformance requirements associated with the **required-character-set**. A **conforming-implementation** must publish, as part of its required documentation, a table giving the correspondence between the graphic symbols in **Table 1** and the **atomic-vector**. Where the graphic symbols provided are not similar in appearance to those used in this standard, the correspondence between the graphic symbols in this standard and those provided by the implementation must also be provided.*

*Some of the graphic symbols given in **Table 1** are not used to designate APL constructs in this standard. They are present to provide all keys on common terminal keyboards with corresponding symbols.*

Table 1 — The Required Character Set

Alphabetic Characters		
<i>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</i>		
<i><u>A</u> <u>B</u> <u>C</u> <u>D</u> <u>E</u> <u>F</u> <u>G</u> <u>H</u> <u>I</u> <u>J</u> <u>K</u> <u>L</u> <u>M</u> <u>N</u> <u>O</u> <u>P</u> <u>Q</u> <u>R</u> <u>S</u> <u>T</u> <u>U</u> <u>V</u> <u>W</u> <u>X</u> <u>Y</u> <u>Z</u></i>		
Numeric Characters		
0 1 2 3 4 5 6 7 8 9		
Special Characters		
α alpha	∇ del tilde	␣ quote quad
↓ down arrow	Δ delta	ρ rho
← left arrow	Δ delta stile	; semicolon
→ right arrow	<u>Δ</u> delta underbar	␣ down shoe
↑ up arrow	¨ diaeresis	␣ left shoe
- bar	◇ diamond	␣ right shoe
blank	÷ divide	␣ up shoe
{ left brace	\$ dollar sign	␣ up shoe jot
} right brace	. dot	/ slash
[left bracket	ε epsilon	\ back slash
] right bracket	= equal	/ slash bar
∨ down caret	≥ greater-than or equal	\ back slash bar
∇ down caret tilde	ι iota	* star
< left caret	◦ jot	stile
> right caret	≤ less-than or equal	␣ down stile
^ up caret	× multiply	␣ up stile
∇ up caret tilde	≠ not equal	␣ down tack
○ circle	— overbar	␣ down tack jot
␣ circle backslash	(left parenthesis	␣ left tack
␣ circle bar	) right parenthesis	␣ right tack
⊗ circle star	+ plus	␣ up tack
␣ circle stile	␣ quad	␣ up tack jot
: colon	␣ quad divide	~ tilde
, comma	? query	_ underbar
∇ del	' quote	ω omega
∇ del stile	! quote dot	

The names in **Table 1** are not to be considered part of this standard.

5.2 NUMBERS

Number-Set: An implementation-defined finite set used to represent arithmetic quantities.

Number: A member of the **number-set**.

Note: The number-set is an abstraction used in this document to represent the floating-point arithmetic quantities of an arbitrary computer.

5.2.1 Elementary Operations

Elementary-Operation: One of the following four implementation-algorithms.

A Plus B.

Note: Plus implements addition.

A Minus B.

Note: Minus implements subtraction.

A Times B.

Note: Times implements multiplication.

A Divided-by B.

Note: Divided-by implements division.

Note: Each elementary-operation is a mapping from the Cartesian product of the number-set with itself back onto the union of the number-set and the metaclass error, as described in the subsection Implementation-Algorithms. It is assumed that members of the metaclass error are returned by elementary-operations when results cannot be represented as numbers. In particular, exponent-overflow is returned when the result of an elementary-operation is too large in magnitude to be represented by a number, exponent-underflow is returned when the result of an elementary-operation is non-zero but too small in magnitude to be represented by a number other than zero, and domain-error is returned by an elementary-operation called with arguments for which its mathematical counterpart is undefined, such as one divided-by zero.

The treatment of exponent-overflow and exponent-underflow is not specified by this standard, except as suggestions to the implementer. Implementers should avoid having exponent-overflow and exponent-underflow occur in the intermediate results of implementation-algorithms wherever possible. In any case, exponent-underflow should not cause a limit-error.

The fact that these fundamental algorithms are effectively undefined in the standard is intentional. A definition general enough to cover all known and possible floating-point systems was felt to be less useful than the requirement that the internal representation for numbers in a conforming-implementation be described in the required-documentation for the implementation.

5.2.2 Number Constants

*Note: The following numbers, used in this standard, are defined here as terms to make clear the distinction between APL constant arrays represented as strings of APL digits, such as **1**, and members of the the implementation-defined finite set containing numbers, such as **one**.*

*These terms are always set in **bold** when they are used in this formal sense. They are not cross-indexed.*

Zero: A number such that, for any number A , A plus zero is A .

One: A number such that, for any number A , A times one is A .

Negative-One: Zero minus one.

Two: One plus one.

Three: Two plus one.

Four: Three plus one.

One-half: One divided-by two.

5.2.3 Subsets of the Set of Numbers

Note: The following subsets of the numbers are used to define the domains of operations.

Boolean: zero or one.

Note: False is represented by zero, true by one.

Note: A counting-number is a member of the subset of the numbers accessible under a successor function.

Positive-Counting-Number: one or any number that can be generated by A plus one, where A is a positive-counting-number.

Negative-Counting-Number: The number negative-one or any number that can be generated by N plus negative-one, where N is a negative-counting-number.

Nonnegative-Counting-Number: A positive-counting-number or zero.

Counting-Number: A negative-counting-number or a nonnegative-counting-number.

Note: An integer is a member of the subset of the numbers accessible from zero and one through addition and subtraction.

Positive-Integer: A positive-counting-number or a number that can be generated by A plus B , where A and B are positive-integers.

Negative-Integer: A negative-counting-number or a number that can be generated by A plus B , where A and B are negative-integers.

Nonnegative-Integer: A positive-integer or zero.

Integer: A negative-integer or a nonnegative-integer.

Positive-Number: A positive-integer, or a number other than zero that can be generated by A times B , by A divided-by B , or by A plus B , where A and B are positive-numbers.

Negative-Number: A negative-integer, or a number other than zero that can be generated by A times B , by A divided-by B , or by C plus D , where A is a positive-number and B , C and D are negative-numbers.

Nonnegative-Number: A positive-number or zero.

Note: The numbers are assumed to comprise the negative-numbers and the nonnegative-numbers.

Example

Consider a normalised base-2 floating point number system with a two position exponent field and a two position mantissa field with the radix point between the digits. The decimal values representable with this system and the categories into which each falls are as follows:

Exponent in Base 2	Mantissa in Base 2	Value in Base 10	Boolean	Counting- Number	Integer
00	00	0	Y	Y	Y
00	10	1	Y	Y	Y
00	11	1.5			
01	10	2		Y	Y
01	11	3		Y	Y
10	10	4		Y	Y
10	11	6			Y
11	10	8			Y
11	11	12			Y

As can be seen in this example, **counting-numbers** form a dense subset of the **integers**; the largest **counting-number** is typically a power of the number system base.

A given number, such as a **counting-number**, may have several hardware representations. Except for their effect on system resources, these representations should be indistinguishable to a **conforming-program**.

5.2.4 Implementation Algorithms

An **Implementation-Algorithm** is an algorithm used in this standard whose behaviour is **implementation-defined**. The following implementation algorithms are used in this standard.

Cosine
Current-Time
Deal
Display
Divided-by
Exponential
Function-Display
Gamma-Function
Hyperbolic-Cosine
Hyperbolic-Sine
Hyperbolic-Tangent
Inverse-Cosine
Inverse-Hyperbolic-Cosine
Inverse-Hyperbolic-Sine
Inverse-Hyperbolic-Tangent
Inverse-Sine
Inverse-Tangent
Matrix-Divide
Minus
Modulo
Natural-Logarithm
Next-Definition-Line
Numeric-Input-Conversion
Numeric-Output-Conversion
Pi-Times
Plus
Pseudorandom-Number-Generator
Read-Keyboard
Sine
Tangent
Times
Time-Stamp
To-the-Power
Trace-Display

The following **implementation-algorithms** take a **number** as an argument and return either a **number** or an error: **Cosine Exponential Gamma-Function Hyperbolic-Cosine Hyperbolic-Sine Hyperbolic-Tangent Inverse-Cosine Inverse-Hyperbolic-Cosine Inverse-Hyperbolic-Sine Inverse-Hyperbolic-Tangent Inverse-Sine Inverse-Tangent Natural-Logarithm Pi-Times Sine Tangent**.

The following **implementation-algorithms** take two **numbers** as arguments and return either a **number** or an error: **Divided-by Minus Modulo Plus Times To-the-Power**.

The properties of the remaining **implementation-algorithms** are described in the chapters in which they are used.

5.2.5 Defined Operations

A Equals B : An operation that, for A and B numbers, returns **one** if A and B are the same number, and **zero** otherwise.

Sign of A : An operation that, for A a number, returns **one** if A is a **positive-number**, **zero** if A is **zero**, and **negative-one** if A is a **negative-number**.

A is Greater-Than B : An operation that, for A and B numbers, returns **one** if A is a **positive-number** and B is a **negative-number**, returns **one** if A minus B is a **positive-number**, and returns **zero** otherwise.

A is Less-Than B : An operation that, for A and B numbers, returns B **greater-than** A .

Negation of A : An operation that, for any number A , returns **zero minus** A .

Absolute-Value of A : An operation that, for any number A , returns A if A is a **nonnegative-number**, and the **negation** of A otherwise.

Exact-Floor of A : An operation that, for any number A , returns the highest integer B such that B equals A or B is **less-than** A .

Exact-Ceiling of A : An operation that, for any number A , returns the lowest integer B , such that B equals A or B is **greater-than** A .

Note: The two preceding definitions assume that the highest and lowest numbers are integers (as is in practice the case in computers).

Open-Interval-Between A and B : An operation that, for any numbers A and B , returns a subset of the numbers as follows: if A is not **greater-than** B , the set of all numbers **greater-than** A and **less-than** B ; otherwise, the **open-interval-between** B and A .

Closed-Interval-Between A and B : An operation that, for any numbers A and B , returns a subset of the numbers consisting of A , B , and the **open-interval-between** A and B .

Left-Neighbourhood of A Determined-by B : An operation that, for any numbers A and B , returns the **closed-interval-between** A and A minus the **absolute-value** of (A times B).

Distance-Between A and B : An operation that, for any two numbers A and B , returns the **absolute-value** of A minus B .

A is Tolerantly-Equal to B Within C : An operation that, given three numbers A , B , and C , returns a **Boolean** Z determined as follows:

If A equals B , then Z is **one**.

If the **sign** of A differs from the **sign** of B , then Z is **zero**.

If the **absolute-value** of B is **greater-than** that of A , and the **absolute-value** of A is in the **left-neighbourhood** of the **absolute-value** of B **determined-by** C , then Z is **one**.

If the **absolute-value** of A is **greater-than** that of B , and the **absolute-value** of B is in the **left-neighbourhood** of the **absolute-value** of A **determined-by** C , then Z is **one**.

Otherwise, Z is **zero**.

Tolerant-Floor of A Within B : An operation that, for A a number and B a **nonnegative-number**, returns an integer Z determined as follows:

Let C stand for the integer for which the **distance-between** A and C is the smallest. If there are two such integers, let C be the one that has the larger **absolute-value**.

If C is **less-than** A , then Z is C .

Otherwise,

If C is not **zero**, then Z is C if A is **tolerantly-equal** to C within B , and Z is C minus one otherwise.

If C is **zero**, then Z is **zero** if A is in the **closed-interval-between** the negation of B and **zero**, and **negative-one** otherwise.

A is **Integral-Within** B : An operation that, for two numbers A and B , returns a **Boolean** Z determined as follows:

Let C stand for the **negation** of A .

Z is **one** if the **tolerant-floor** of C within B equals the **negation** of the **tolerant-floor** of A within B , and **zero** otherwise.

A is a **Near-Integer**: An operation that, for a number A , returns **one** if A is **integral-within integer-tolerance**, and **zero** otherwise.

Note: This definition contains a forward reference to integer-tolerance.

Integer-Nearest-to A : An operation that, for a **near-integer** A , returns the **tolerant-floor** of A within **integer-tolerance**.

A is **Near-Boolean**: An operation that, for a **near-integer** A , returns **one** if the **integer-nearest-to** A is a **Boolean**, and **zero** otherwise.

Note: Note that near-integers and near-Booleans include numbers whose absolute-value is less-than integer-tolerance. The operation integer-nearest-to maps such numbers to zero.

5.3 OBJECTS

Note: The description of APL in this document is based on objects – abstract data structures that are described in terms of characters, numbers, and elementary set theory.

Each object has a small set of named attributes that take on values that are characters, numbers, or other objects.

Each object has additionally a small set of operations that can be performed upon it.

All the objects in the description are defined here, as are all the attributes of each object. The operations are introduced as they are needed. All objects, attributes, and operations are cross-referenced in the index.

5.3.1 Lists

Index: A **nonnegative-counting-number** less than or equal to **index-limit**.

List: An object with the following attributes:

Index-Set: A finite set I of **positive-counting-numbers** chosen so that for every subset of I , except the empty set, the cardinality of that subset is in I .

Value-Set: A finite set in a specific correspondence with the **index-set** of the list.

Empty-List: A list, the cardinality of whose **index-set** is **zero**.

Nonempty-List: A list, the cardinality of whose **index-set** is **greater-than zero**.

Number-of-Items in L : The cardinality of the **index-set** of the list L .

Item X of L : An operation that, for X a member of the **index-set** of the list L , returns the member of the **value-set** of L associated with X by the correspondence between the **value-set** of L and the **index-set** of L .

Note: Note that this form of indexing is always in origin one.

First-Item in L : An operation that for a **nonempty-list** L returns **item one** of L .

Last-Item in L : An operation that for a **nonempty-list** L , and for C the **number-of-items** in L , returns **item** C of L .

Rest-of L : An operation that, for a **nonempty-list** L whose **index-set** has cardinality C , returns a second list R whose **index-set** has cardinality C **minus one**, such that for each **item** J in the **index-set** of R , **item** J of R is **item** $(J$ **plus one**) of L .

Product-of L : An operation defined on a list of numbers L as follows:

If the **number-of-items** in L is **zero**, **one**.

If the **number-of-items** in L is **one**, the **first-item** in L .

If the **number-of-items** in L is **greater-than one**, **first-item** in L times the **product-of** the **rest-of** L .

Prefix: A (possibly empty) list P is a **prefix** of a list L if the **index-set** of P is a subset of the **index-set** of L , and each **item** of P is the same as the corresponding **item** of L .

5.3.2 Arrays

Array-Type: An enumerated set containing the members **character** and **numeric**.

Array: An object with the following attributes:

Shape-list: A list of **nonnegative-counting-numbers**.

Ravel-list: A list. The items of the list may be either **characters** or **numbers**.

Type: A member of the enumerated set **array-type**.

The **number-of-items** in the **ravel-list** of an array A is the same as the **product-of** the **shape-list** of A .

If the **type** of A is **character**, the **ravel-list** of A is a list of **characters**; if the **type** of A is **numeric**, the **ravel-list** of A is a list of **numbers**.

Rank of A : An operation that, for an array or array-of-vectors A , returns the **number-of-items** in the **shape-list** of A .

Scalar: An array whose **rank** is **zero**.

Numeric-Scalar with value I : For I , a **number**, the **scalar** Z such that the **type** of Z is **numeric** and the **ravel-list** of Z is the list L such that the **number-of-items** in L is **one** and the **first-item** of L is I .

Count of A : For A an array, the **product-of** the **shape-list** of A .

Vector: An array whose **rank** is **one**.

Length of A : For A a **vector**, the **count** of A .

Typical-Element of A : For A an array, if the **type** of A is **character**, the **character** blank; if the **type** of A is **numeric**, **zero**.

Empty: Of an array A , the **count** of A is **zero**.

One-Element-Vector: A **vector** A is a **one-element-vector** if the **length** of A is **one**.

K is a Valid-Axis of A : An operation that, for A and K arrays, returns **one** if K is a **scalar** or **one-element-vector**, and the **first-item** in the **ravel-list** of K is a **near-integer**, the **integer-nearest-to** which is a member of the **index-set** of the **shape-list** of A ; and returns **zero** otherwise.

Axis K of A : An operation that, for A an array and K a **valid-axis** of A , returns **item** K of the **shape-list** of A .

Array-of-vectors: An object with the following attributes:

Shape-list: A list of **nonnegative-counting-numbers**.

Ravel-list: A list. The items of the list are **vectors**, all of the same **type**.

Type: A member of the enumerated set **array-type**.

Along-Axis K of A : An operation that, for an array A with non-zero **rank** N , produces Z , an **array-of-vectors** of **rank** $N - 1$ such that the **shape-list** of Z is the **shape-list** of A with **item** K omitted. Each **item** of the **ravel-list** of Z is a **vector** whose **length** is the same as **axis** K of A .

Note: Some operations on vectors extend to arrays of greater rank in a manner similar to scalar-extension. Along-axis is an expository device used in the definition of these operations.

Vector-Item I of A : An operation that, for an **array-of-vectors** A , returns **item** I of the **ravel-list** of A .

Ravel-Along-Axis K of A : An operation that, for an **array** A with non-zero **rank** N , produces an **array-of-vectors** Z such that the **shape-list** of Z is the **product-of** the **shape-list** of A with **item** K omitted, and the **ravel-list** of Z is the **ravel-list of along-axis** K of A .

First-Scalar in A : An operation that, for A a nonempty **array**, returns a **scalar** Z such that the **type** of Z is the **type** of A and the **ravel-list** of Z is the **first-item** in the **ravel-list** of A .

Remainder-of A : An operation that, for A a non-empty **vector**, returns a **vector** Z such that the **type** of Z is the **type** of A , the **length** of Z is **negative-one plus** the **length** of A , and the **ravel-list** of Z is the **rest-of** the **ravel-list** of A .

Row I of A : An operation that, for A an **array of rank two** and I a **number**, returns **vector-item** I **along-axis two** of A .

Number-of-Rows in A : An operation that, for A an **array of rank two**, returns **item one** of the **shape-list** of A .

Integer-Array-Nearest-to A : An operation that, for A an **array** having the property that each **item** of the **ravel-list** of A is a **near-integer**, returns a **numeric array** Z such that the **shape-list** of Z is the same as the **shape-list** of A and each **item** of the **ravel-list** of Z is the **integer-nearest-to** the corresponding **item** of the **ravel-list** of A .

Boolean-Array-Nearest-to A : An operation that, for A an **array** having the property that each **item** of the **ravel-list** of A is a **near-Boolean**, returns the **integer-array-nearest-to** A .

5.3.3 Defined-Functions

Note: A **defined-function** represents a function defined by a user. The attributes of a **defined-function** are given here. Operations are described in the **Defined Functions** chapter.

Defined-Function: An object with the following attributes:

Canonical-Representation: A character array whose rank is two.

Stop-Vector: A numeric vector.

Trace-Vector: A numeric vector.

5.3.4 Tokens

Class-Names: A set containing the following members:

Assignment-Arrow	Local-Name
Axis-Error	Monadic-Operator
Branch	Nil
Branch-Arrow	Niladic-Defined-Function
Character-Literal	Niladic-Defined-Function-Name
Clear-State-Indicator	Niladic-System-Function-Name
Colon	Not-Copied
Command-Complete	Not-Erased
Committed-Value	Not-Found
Complete-Index-List	Not-Saved
Constant	Numeric-Literal
Defined-Function	Partial-Index-List
Defined-Function-Name	Primitive
Definition-Error	Primitive-Function
Distinguished-Identifier	Rank-Error
Domain-Error	Result-Name
Dyadic-Operator	Right-Argument-Name
Elided-Index-Marker	Right-Axis-Bracket
Escape	Right-End-of-Statement
Implicit-Error	Right-Index-Bracket
Incorrect-Command	Right-Parenthesis
Index-Error	Semicolon
Index-Separator	Shared-Variable
Interrupt	Shared-Variable-Name
Label	Simple-Identifier
Label-Name	Small-Circle
Left-Argument-Name	Syntax-Error
Left-Axis-Bracket	System-Function-Name
Left-End-of-Statement	System-Variable-Name
Left-Index-Bracket	Unwind
Left-Parenthesis	Value-Error
Length-Error	Variable
Limit-Error	Variable-Name

Token: An object with the following attributes:

Class: A member of the set **Class-Names**.

Content: A number, a character, or an object, according to the class of the token, as indicated in Table 2.

Note: Tokens represent the internal objects manipulated by an implementation of APL.

The class of a token is a straightforward indication of the sort of object it represents. A token of class nil, for example, is used to return a value-error from a defined function that does not set its result name. Note that class-names is an enumerated set. This means that nil has no definition other than its literal appearance on this page. Its significance, like the significance of APL characters in this document, lies in the use of its name in evaluation sequences.

The content of a token varies with the class of the token. If specified, the content is a character, a list of characters, a list of numbers, an array, an index-list, a shared-variable, or a defined-function.

Table 2 — Relationship between Class-Name and Content

Class-Name	Content	Class-Name	Content
Assignment-Arrow		Local-Name	A list of characters
Axis-Error		Monadic-Operator	A character
Branch	An array	Nil	
Branch-Arrow		Niladic-Defined-Function	A defined-function
Character-Literal	A list of characters	Niladic-Defined-Function-Name	A list of characters
Clear-State-Indicator		Niladic-System-Function-Name	A list of characters
Colon		Not-Copied	
Command-Complete		Not-Erased	
Committed-Value	An array	Not-Found	
Complete-Index-List	An index-list	Not-Saved	
Constant	An array	Numeric-Literal	A list of numbers
Defined-Function	A defined-function	Partial-Index-List	An index-list
Defined-Function-Name	A list of characters	Primitive	A character
Definition-Error		Primitive-Function	A character
Distinguished-Identifier	A list of characters	Rank-Error	
Domain-Error		Result-Name	A list of characters
Dyadic-Operator	A character	Right-Argument-Name	A list of characters
Elided-Index-Marker		Right-Axis-Bracket	
Escape		Right-End-of-Statement	
Implicit-Error		Right-Index-Bracket	
Incorrect-Command		Right-Parenthesis	
Index-Error		Semicolon	
Index-Separator		Shared-Variable	A shared-variable
Interrupt		Shared-Variable-Name	A list of characters
Label	An array	Simple-Identifier	A list of characters
Label-Name	A list of characters	Small-Circle	
Left-Argument-Name	A list of characters	Syntax-Error	
Left-Axis-Bracket		System-Function-Name	A list of characters
Left-End-of-Statement		System-Variable-Name	A list of characters
Left-Index-Bracket		Unwind	
Left-Parenthesis		Value-Error	
Length-Error		Variable	An array
Limit-Error		Variable-Name	A list of characters

5.3.4.1 Metaclasses

*Note: Metaclasses are subsets of the enumerated set **class-names**. They are used to shorten evaluation sequences by abstracting a sequence of tests on the **class** of a **token** into a single test for membership in a metaclass. "If *B* is an **error**," is equivalent to "If the **class** of *B* is in the metaclass **error**."*

Metaclass: A subset of the enumerated set **class-names**.

Identifier: A metaclass containing **simple-identifier** and **distinguished-identifier**.

Literal: A metaclass containing **character-literal** and **numeric-literal**.

Lexical-Unit: A metaclass containing **primitive**, **literal**, and **identifier**.

Value: A metaclass containing **committed-value** and **constant**.

Delimiter: A metaclass containing **primitive-function**, **branch-arrow**, **assignment-arrow**, **left-end-of-statement**, **right-end-of-statement**, **left-index-bracket**, **right-index-bracket**, **elided-index-marker**, **left-axis-bracket**, **right-axis-bracket**, **left-parenthesis**, **right-parenthesis**, **small-circle**, and **semicolon**.

Defined-Name: A metaclass containing **shared-variable-name**, **variable-name**, **defined-function-name**, **niladic-defined-function-name**, **result-name**, **left-argument-name**, **right-argument-name**, **label-name**, and **local-name**.

System-Name: A metaclass containing **system-variable-name**, **system-function-name**, and **niladic-system-function-name**.

Classified-Name: A metaclass containing the members of **system-name** and **defined-name**.

Syntactic-Unit: A metaclass containing the members of **classified-name** and **delimiter**.

Error: A metaclass containing **axis-error**, **domain-error**, **implicit-error**, **index-error**, **length-error**, **limit-error**, **rank-error**, **syntax-error**, **value-error**, and **interrupt**.

Note: This metaclass includes only errors that occur in the evaluation of APL statements.

Report: A metaclass containing **incorrect-command**, **not-copied**, **not-erased**, **not-found**, and **not-saved**.

Exception: A metaclass containing **branch**, **escape**, **clear-state-indicator**, **unwind**, and the members of **error** and **report**.

Result: A metaclass containing **nil** and the members of **exception** and **value**.

5.3.4.2 Index-List

Index-List: A (possibly empty) **list** consisting of **tokens** whose **class** is either **constant** or **elided-index-marker**.

5.3.5 Symbols

Symbol: An object with the following attributes:

Name: A **list** of **characters**.

Referent-List: A **list** of **tokens**.

5.3.6 Contexts

Mode-Names: An enumerated set containing the members **immediate-execution**, **execute**, **function-definition**, **quad-input**, and **defined-function**.

Context: An object with the following attributes:

Mode: An element of the enumerated set **mode-names**.

Stack: A list of **tokens**.

Current-Line: A list of **characters**.

Current-Statement: A list of **tokens**.

Current-Function: If **mode** is **defined-function**, a **defined-function**; otherwise undefined.

Current-Line-Number: If **mode** is **defined-function**, an **index**; otherwise undefined.

5.3.7 Workspaces

Note: The workspace is the basic organisational unit in an APL system. A workspace contains data, programs, execution status, and environmental information.

Workspace-Presence: An enumerated set containing **absent** and **present**.

Workspace: An object with the following attributes:

Owner: A **user-identification**.

Workspace-Name: A list of **characters**.

Symbol-Table: A list of all **symbols** whose **names** are unique.

State-Indicator: A list of **contexts**.

Existential-Property: A member of the enumerated set **workspace-presence**.

Clear-Workspace: A **workspace** with the following values:

Owner: **this-owner**.

Workspace-Name: The **clear-workspace-identifier**.

Symbol-Table: A list of all **symbols** whose **names** are unique and whose **referent-lists** each consist of the **list** whose only member is **nil**.

State-Indicator: An empty list of **contexts**.

Existential-Property: **absent**.

5.3.8 Sessions

Note: A user interacts with an APL system through a session, an abstraction that represents a hypothetical machine capable of carrying out the evaluation sequences in the standard.

The attributes of a session are given here.

Session-Identification: Either a number or a list of characters, depending upon the implementation-parameter session-identification-type. **User-Identification:** Either a number or a list of characters, depending upon the implementation-parameter user-identification-type.

Keyboard-States: An enumerated set containing the members open-keyboard and locked-keyboard.

Session: An object with the following attributes:

Active-Workspace: A workspace.

This-Session: A session-identification.

This-Owner: A user-identification.

Attention-Flag: A member of the enumerated set Boolean.

Keyboard-State: A member of the enumerated set keyboard-states.

Current-Prompt: A character vector.

Quote-Quad-Prompt: A character vector.

Current-Context: The first-item in the state-indicator of the active-workspace.

Current-Stack: The stack of the current-context.

Symbol-Named-By T : The symbol in the symbol-table of the active-workspace whose name is the same as the content of the token T .

Current-Referent of T : The first-item in the referent-list of the symbol-named-by T .

Current-Class of T : The class of the current-referent of T .

Current-Content of T : The content of the current-referent of T .

Comparison-Tolerance: The current-content of $\square CT$.

Random-Link: The current-content of $\square RL$.

Print-Precision: The current-content of $\square PP$.

Index-Origin: The current-content of $\square IO$.

Latent-Expression: The current-content of $\square LX$.

System-Parameter: Any of comparison-tolerance, random-link, print-precision, index-origin, or latent-expression.

The initial values of system-parameters in a clear-workspace are implementation-defined.

5.3.9 Shared-Variables

*Note: A **shared-variable** is a variable shared between two sessions.*

Shared Variables are an **optional-facility**.

Shared-Variable: An object with the following attributes:

Session-A: A session-identification.

Session-A-Active: A Boolean.

Session-A-ACV: A Boolean vector of length four.

Session-B: A session-identification.

Session-B-Active: A Boolean.

Session-B-ACV: A Boolean vector of length four.

Shared-Name: An identifier.

Shared-Value: A token, either a constant or nil.

State: An integer, either zero, one, or two.

*Note: The operations in the chapter **Shared Variables** use the **shared-variable** attributes as follows:*

***Session-A:** The session-identification of the first session to offer the **shared-variable**.*

***Session-A-Active:** A Boolean; one if session-A is currently sharing this **shared-variable**, zero if not.*

***Session-A-ACV:** The contribution of session-A to the **ACV**.*

***Session-B:** The session-identification of the session with which session-A offered to share the **shared-variable**; this may be the **general-offer** while **state** is one.*

***Session-B-Active:** A Boolean; one if session-B is currently sharing this **shared-variable**, zero if not.*

***Session-B-ACV:** The contribution of session-B to the **ACV**.*

***Shared-Name:** An identifier, the name session-A designated for this **shared-variable**.*

***Shared-Value:** A token of class **constant** or nil; the current value of this **shared-variable**.*

***State:** An integer, either zero, one, or two, used in combination with the **ACV** to determine which operations must be delayed.*