

INTERNATIONAL STANDARD

**ISO
8485**

First edition
1989-11-01

Programming languages — APL

Langages de programmation — APL



Reference number
ISO 8485 : 1989 (E)

Foreword

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

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8485 was prepared by Technical Committee ISO/TC 97, *Information processing systems*.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Annexes A and B are for information only.

CONTENTS

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	$Z \leftarrow f\neq B$	114
9.2.1 Reduction	$Z \leftarrow f\neq[K] B$	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	$Z \leftarrow \Psi B$	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	$Z \leftarrow \mathfrak{B} B$	124
10.1.6 Matrix Inverse	$Z \leftarrow \mathfrak{B} B$	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	$Z \leftarrow A \neq B$	133
10.2.5 Compress	$Z \leftarrow A /[K] B$	133
10.2.5 Compress	$Z \leftarrow A \neq[K] B$	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	$Z \leftarrow A \mathfrak{B} B$	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	$Z \leftarrow A \mathfrak{B} B$	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

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

2 REFERENCES

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

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

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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 $Z \leftarrow f \setminus B$.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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.*

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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

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

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	Δ 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

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.

5.3.10 Systems

Note: A system represents a set of active APL users, a library, and, optionally, a shared variable facility.

System: An object with the following attributes:

Library: A list of **workspaces** in which each combination of possible values for the attributes **owner** and **workspace-name** occurs exactly once.

Active-Users: A list of **sessions**.

Shared-Variable-List: A list of **shared-variables**.

Implementation-Parameters: The following quantities, referred to in this standard by name, whose values are **implementation-defined**:

Atomic-Vector: An **implementation-defined** character vector containing every member of the **character-set** exactly once.

Initial-Comparison-Tolerance: A member of the **internal-value-set** for **comparison-tolerance** that is the value of **comparison-tolerance** in a **clear-workspace**.

Initial-Index-Origin: A member of the **internal-value-set** for **index-origin** that is the value of **index-origin** in a **clear-workspace**.

Initial-Latent-Expression: A member of the **internal-value-set** for **latent-expression** that is the value of **latent-expression** in a **clear-workspace**.

Initial-Print-Precision: A member of the **internal-value-set** for **print-precision** that is the value of **print-precision** in a **clear-workspace**.

Initial-Random-Link: A member of the **internal-value-set** for **random-link** that is the value of **random-link** in a **clear-workspace**.

Clear-Workspace-Identifier: A list of characters.

Positive-Number-Limit: The number **greater-than** all other numbers.

Negative-Number-Limit: The number **less-than** all other numbers.

Positive-Counting-Number-Limit: The **counting-number** greater than all other **counting-numbers**.

Negative-Counting-Number-Limit: The **counting-number** **less-than** all other **counting-numbers**.

Index-Limit: The **index** greater than all other indices. This value specifies the maximum value of any **item** of the **shape-list** of any **array**, ignoring storage limitations.

Count-Limit: An **index** not greater than **index-limit** that specifies the maximum value for the **count** of an array, ignoring storage limitations.

Rank-Limit: An **index** not greater than **count-limit** that specifies the maximum value for the **rank** of any array, ignoring storage limitations.

Workspace-Name-Length-Limit: A **positive-counting-number** that specifies the maximum number of characters in a **workspace-name**.

Identifier-Length-Limit: A **positive-counting-number** not greater than **count-limit** that specifies the maximum number of characters in an **identifier**.

Quote-Quad-Output-Limit: A **nonnegative-counting-number** that specifies the maximum number of characters that can be used in a prompt set by **Quote Quad Output**.

Comparison-Tolerance-Limit: The largest **number** permitted by the implementation for the system parameter **comparison-tolerance** .

Integer-Tolerance: A **number** not greater than **comparison-tolerance-limit** used to determine whether a given number is to be considered integral.

Full-Print-Precision: The smallest **positive-counting-number** which, when used as the value of **print-precision** , causes **numeric-output-conversion** to produce a unique **numeric-scalar-literal** for every **number**.

Print-Precision-Limit: The largest **positive-counting-number** permitted by the implementation for the system parameter **print-precision** .

Note: Print-precision-limit must be at least full-print-precision if every unique number is to have a unique decimal representation.

Exponent-Field-Width: A **positive-counting-number** giving the number of places, including sign and trailing blanks, used for the exponent field in the result of **dyadic format**.

Session-Identification-Type: A member of the enumerated set **array-type**, either **character** or **numeric**.

User-Identification-Type: A member of the enumerated set **array-type**, either **character** or **numeric**.

Indent-Prompt: A list of **characters** used to indicate to the user that the **session** is in **immediate-execution** mode.

Quad-Prompt: A list of **characters** used to indicate to the user that the **session** is in **quad-input** mode.

Function-Definition-Prompt: A list of **characters** used to indicate to the user that the **session** is in **function-definition** mode.

Line-Limit: A **positive-counting-number** that specifies the maximum number of lines permitted in a defined function, ignoring storage limitations.

Definition-Line-Limit: A **positive-number** that specifies the maximum value of a line number in **function-definition** mode.

General-Offer: A reserved **session-identification** used to indicate that the offerer of a **shared-variable** is willing to share the proffered variable with any other **session**. This **implementation-parameter** is required only if the **optional-facility shared-variable-protocol** is present.

Any action that would cause a limit specified by an implementation parameter to be exceeded shall signal a **limit-error**.

5.4 EVALUATION SEQUENCES

The evaluation sequences that define APL operations in the remainder of this standard are written in English in the imperative mood. The English phrases used in evaluation sequences are restricted to the set specified in this subsection.

Indentation is used in evaluation sequences to indicate scope, typically of the consequent of an implication.

*For example, in the evaluation sequence fragment below, the indented text is evaluated only if both A and B are **vectors**; the "otherwise" clause is evaluated only if at least one of A or B is not a **vector**.*

*If A is a **vector** and B is a **vector**,*

*If A is **empty** and ...*

Otherwise, return ...

Expressions in APL are used in evaluation sequences. A given evaluation sequence uses only APL operations that have been specified earlier in the standard. Where indices are generated or used by APL expressions in evaluation sequences, they are evaluated with origin **one**. The APL relational operations are never used in evaluation sequences unless they are qualified with the value to be used for **comparison-tolerance**.

5.4.1 Evaluation Sequence Phrases

Note: The following phrases are used in evaluation sequences. They are not set in **bold** type nor cross-referenced when employed in evaluation sequences.

For all A, C : An evaluation sequence phrase used to specify that the action or condition specified by the consequent C is to be performed or checked for every value in the antecedent A .

Example:

*For all I in the **index-set** of A ,*

*Set item I of the **ravel-list** of A to **zero**.*

For form F, C :

For pattern F, C :

An evaluation sequence phrase used to specify that the actions listed in the consequent C are to be performed only if the pattern or form F is the one that caused this evaluation sequence to be selected.

Example:

For form $A \phi B$

If B is a **scalar** ...

For form $A \Theta B$

If ...

If A, C : An evaluation sequence phrase used to specify that the actions listed in the consequent C are to be performed only if the value of the antecedent A is **one**.

If T is an **mc**, C : For T a **token** and **mc** a **metaclass**, an evaluation sequence phrase equivalent to "If the **class** of T is in the **metaclass** **mc**, C ."

Let A **stand for** B : An evaluation sequence phrase used to indicate that the name A is to be an abbreviation for the phrase B in subsequent evaluation sequence lines.

Otherwise, C : An evaluation sequence phrase used to indicate that the actions listed in consequent C are to be performed only if the antecedent in the immediately preceding if phrase was **zero**. If a consequent is an indented paragraph, the immediately preceding if statement is the one at the same level of indentation as the **otherwise** phrase.

Repeat: An evaluation sequence phrase used to indicate that the block of text indented after the **repeat** is to be executed repeatedly until a **return** or **signal** phrase is encountered. The end of a repeated block is indicated by the parenthetic remark "(End of repeated block)."

Return X : An evaluation sequence phrase used to specify that evaluation of this evaluation sequence is to stop and that a **token** is to be returned to the caller of the evaluation sequence. If X is a **token**, then X is returned; if X is an **array**, a **token** of class **constant** and content X is returned.

Set A **to** B : An evaluation sequence phrase used to specify that the referent of A is to be assigned the value B .

Signal X : An evaluation sequence phrase used to specify that evaluation of this evaluation sequence is to stop and that a **token** whose **class** is X , where X is an **error**, is to be returned.

Using O, C : An evaluation sequence phrase used to indicate that the consequent C is to be evaluated against the specific object O . This construct is used, for example, in the description of shared variables to indicate which shared variable is to be changed.

Wait until: An evaluation sequence phrase that indicates that the **session** is waiting for a condition to hold before continuing.

5.4.2 Diagrams

Note: *Diagrams are used in this standard to indicate permissible sequences of **characters** or of **tokens**.*

Character-Diagram: A graph that designates a subset of the set of all **lists** of **characters**.

Token-Diagram: A graph that designates a subset of the set of all **lists** of **tokens**.

Thread D with A : An evaluation sequence phrase used to indicate that a search is to be made for a route through the diagram D that corresponds to the list or part of a list A .

A **character-diagram** is **threaded** with a **list** of **characters** by setting a list-cursor to the first item in the **list**, and a diagram-cursor to the arrow-tail ' $\rightarrow$ ', in the diagram, then progressing along paths in the diagram to the arrow-head ' $\rightarrow$ '. The diagram-cursor advances to the left only through a left rectifier ' $\leftarrow$ '. It advances upward only after it has just passed a right rectifier ' $\rightarrow$ ' or a left rectifier ' $\leftarrow$ '. Otherwise, the diagram-cursor advances only rightward and downward. At a junction, any unlabelled path may be taken only if all labelled paths have already been tried.

For the diagram-cursor to advance along a path labelled with a graphic symbol such as $\div$, that graphic symbol must be pointed to by the list-cursor, which then also advances. For the diagram-cursor to advance along a path labelled with a diagram name, that diagram must itself be threaded, using the same list-cursor and a new diagram-cursor. If in either case the diagram-cursor cannot advance, the diagram-cursor is set back to the previous junction, the list-cursor is set back correspondingly, and a new path is tried; or if the diagram-cursor is now at the arrow-tail, the diagram cannot be threaded with the list.

If a route is found through a diagram for a given **list** of **characters**, it is unique. In this case, the diagram can be **rethreaded**, following the same route without entering blind alleys. For example, characters are collected into identifier tokens through actions performed while a **character-diagram** is being **rethreaded**.

Token-diagrams are **threaded** exactly like **character-diagrams**, except that the items pointed to by the list-cursor are **tokens** and are matched either by their **class** or by both their **class** and their **content**.

Rethread D with A : An evaluation sequence phrase used to indicate that the route found by a previous **threaded** phrase is to be threaded again, in order to effect certain actions through **when** phrases.

When A , C : An evaluation sequence phrase used during a **rethread** phrase to indicate that when the antecedent A is true, the consequent C is to be performed.

A **Matches** D : An operation that, for diagram D and **list** A , is true if A is a member of the set of **lists** designated by D , and false otherwise.

A **list** matches a diagram if it can be **threaded** in such a way that there are no items remaining in the **list** when the final exit path is taken.

Note: The question of whether a **list** **matches** a diagram is different from the question of whether a **list** can **thread** a diagram, since there will normally be items left over in a **list** once a diagram has been **threaded** — in collecting the digits in a number, for example, the **digit** diagram removes only one digit at a time from a **list** of **characters**.

5.5 OTHER TERMS

Side-effect: Any effect an operation has other than returning a result.

Atomic: The property of an operation with **side-effects** to produce its **side-effects** only if it completes without error.

Note: For example, the specification of items 3, 4, and 5 of $\bar{A}$ in the APL line

$B \leftarrow A[3 \ 4 \ 5] \leftarrow 2$

is a side effect, since the result placed in B is the scalar 2; further, since **indexed assignment** is specified to be **atomic**, no change to $\bar{A}$ will occur if the indexed assignment fails with, for example, an index error.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

6 SYNTAX AND EVALUATION

6.1 INTRODUCTION

Note: This chapter specifies the rules for evaluating lines. These rules are used by the subsections **execute**, **quad input**, **immediate-execution**, and **defined-function-control**.

The rules are described in three subsections, **evaluate-line**, **evaluate-statement**, and **reduce-statement**.

The data structures and procedures used to describe syntax and evaluation in this standard are strictly expository devices; they are not intended to represent required or desirable implementation techniques. The order of reporting errors implied by the diagram-matching actions in the model is not required.

6.1.1 Evaluate-Line

Form: **Evaluate-Line**

Informal Description: **Evaluate-line** is the principal procedure in the evaluation of APL lines. It decomposes the list of **characters** named **current-line** into **lexical-units** according to the **character-diagram** named **line**.

Some diagrams referred to in **line** have their exit paths flagged with asterisks. Once the diagram **line** has been threaded, the rethreading pass is used to gather certain character sequences (diagrams ending in two asterisks), and either create tokens (diagrams ending in one asterisk) or discard the character sequences gathered (diagrams ending in three asterisks).

Evaluate-line calls **evaluate-statement** to convert the **lexical-units** into a **result**.

Note that the **result** can be a **constant** (from, for example, $2+2$), **committed-value** (from $A \leftarrow 1$), **nil** ($\emptyset$ ' '), **branch** ($\emptyset$ ' $\rightarrow 3$ '), **escape** ($\rightarrow$), **unwind** ($\rightarrow$), **error** ($2-$), **clear-state-indicator** (from $)SIC$) or **interrupt** (from **signal-interrupt**).

Note: The handling of these result classes by the different callers of **evaluate-line** specifies the treatment of exceptions, and should be analysed carefully.

Evaluate-line is called from evaluation sequences in the subsections **defined-function-control**, **execute**, **immediate-execution**, and **quad-input**. If the **optional-facility statement-separator-facility** is implemented, **evaluate-line** evaluates successive statements in **current-line** beginning with the leftmost statement and continuing until evaluation produces an **exception** or the rightmost statement has been evaluated.

Evaluation Sequence:

Set C to the **empty-list of characters**.

Set **current-statement** to the **empty-list of tokens**.

Thread **line** with **current-line**; if **current-line** does not match **line**, signal **syntax-error**.

Rethread **line** with **current-line**, taking the following actions:

When a **character-diagram** ending in '**-***>>**' is recognised, set C to the **empty-list of characters**.

When a **character-diagram** ending in '**-**>>**' is recognised, append to C as a new last item the **character** just passed.

When a **character-diagram** ending in '**-*>>**' is recognised,

Append to **current-statement** as a new last item a **token** with **class** given by the name of the **character-diagram** and with **content** C .

Set C to the **empty-list of characters**.

When the **optional-facility statement-separator-facility** is implemented and the **character-diagram statement-separator** is recognised,

Set Z to **evaluate-statement**.

If Z is an **exception**, return Z .

If Z is a **constant**, **display** Z .

Set **current-statement** to the **empty-list of tokens**.

Set C to the **empty-list of characters**.

When the **character-diagram line** is matched,

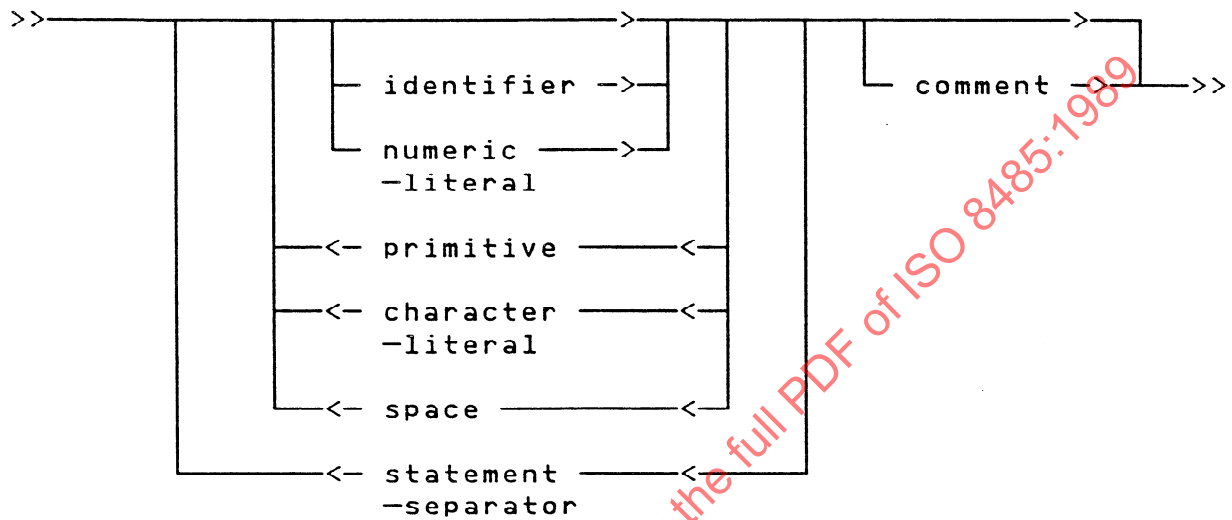
Set Z to **evaluate-statement**.

Return Z .

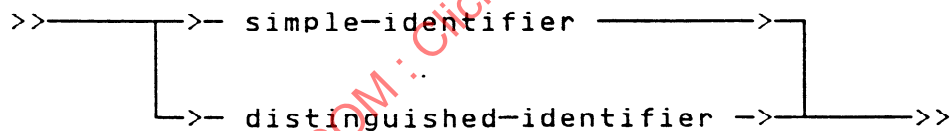
6.1.2 Character-Diagrams

The diagrams in this subsection are **character-diagrams**: the APL Graphic Symbols such as ∇ in these **character-diagrams** match corresponding **characters** in the **required-character-set**.

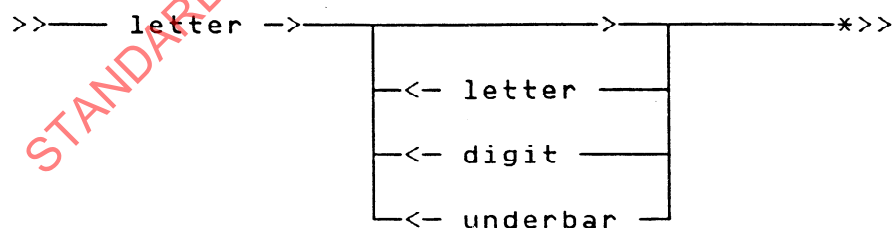
Line



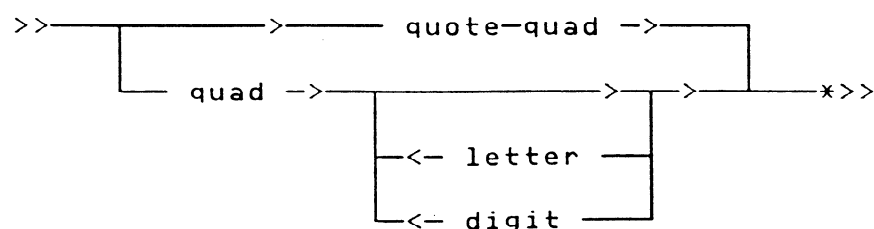
Identifier



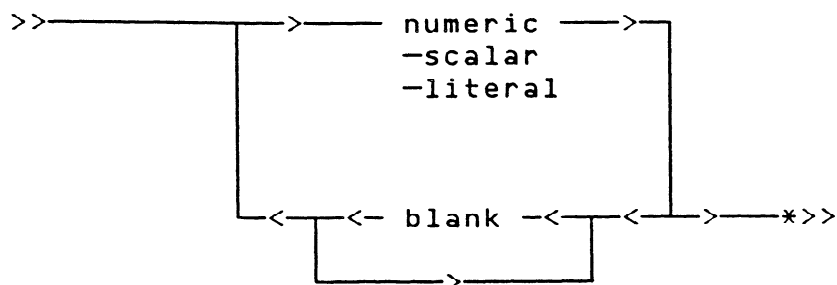
Simple-Identifier



Distinguished-Identifier

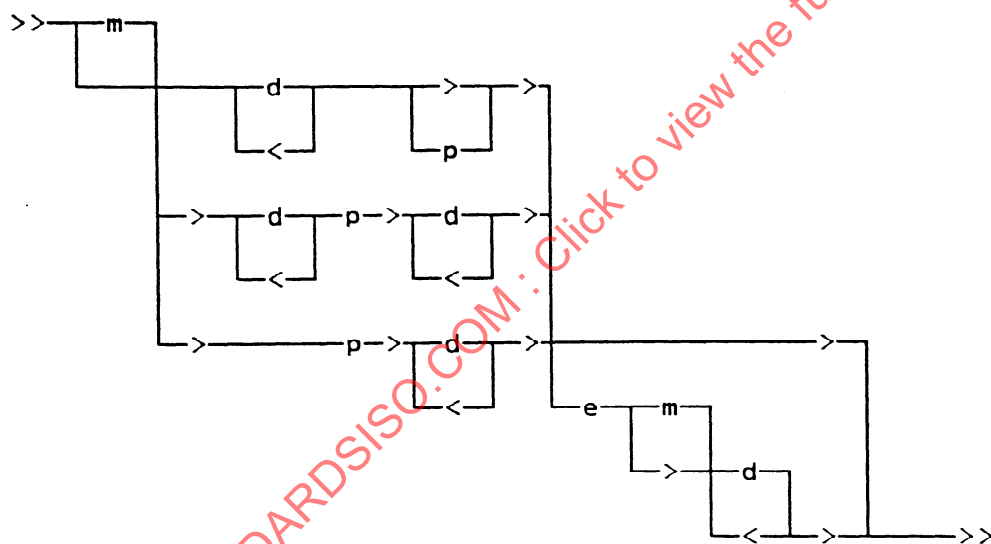


Numeric-Literal



Numeric-Scalar-Literal

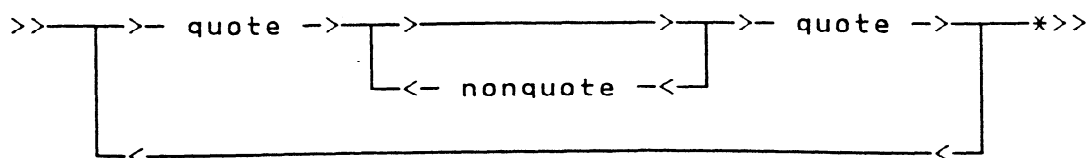
d stands for **digit**.
 e stands for **exponent-marker**.
 m stands for **overbar**.
 p stands for **dot**.

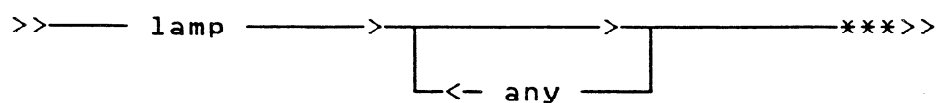
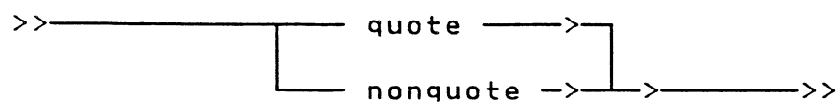
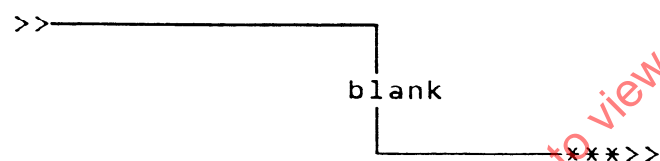
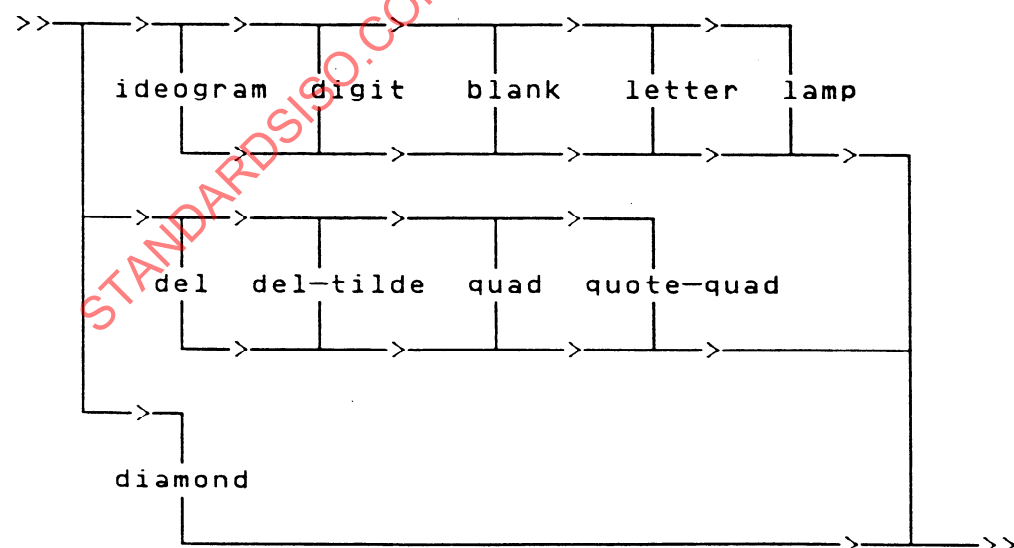


Example:

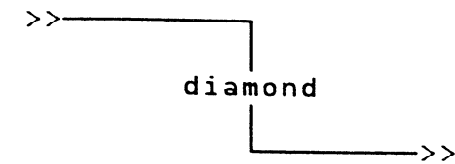
$\overline{12.34567E^{-890}}$

Character-Literal

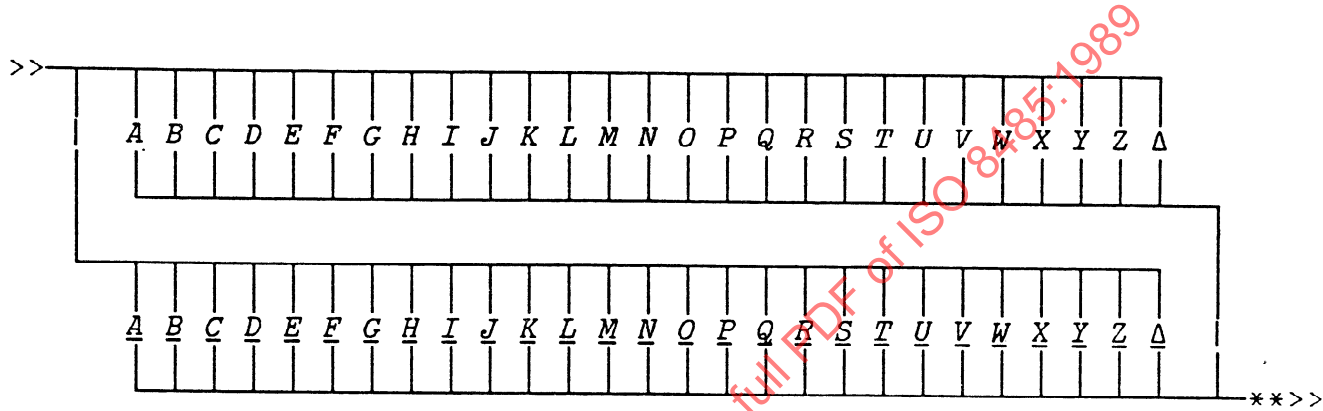


Comment**Any****Primitive****Space****Nonquote**

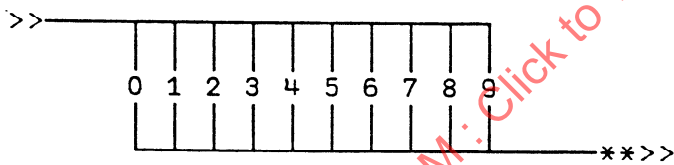
Statement-Separator



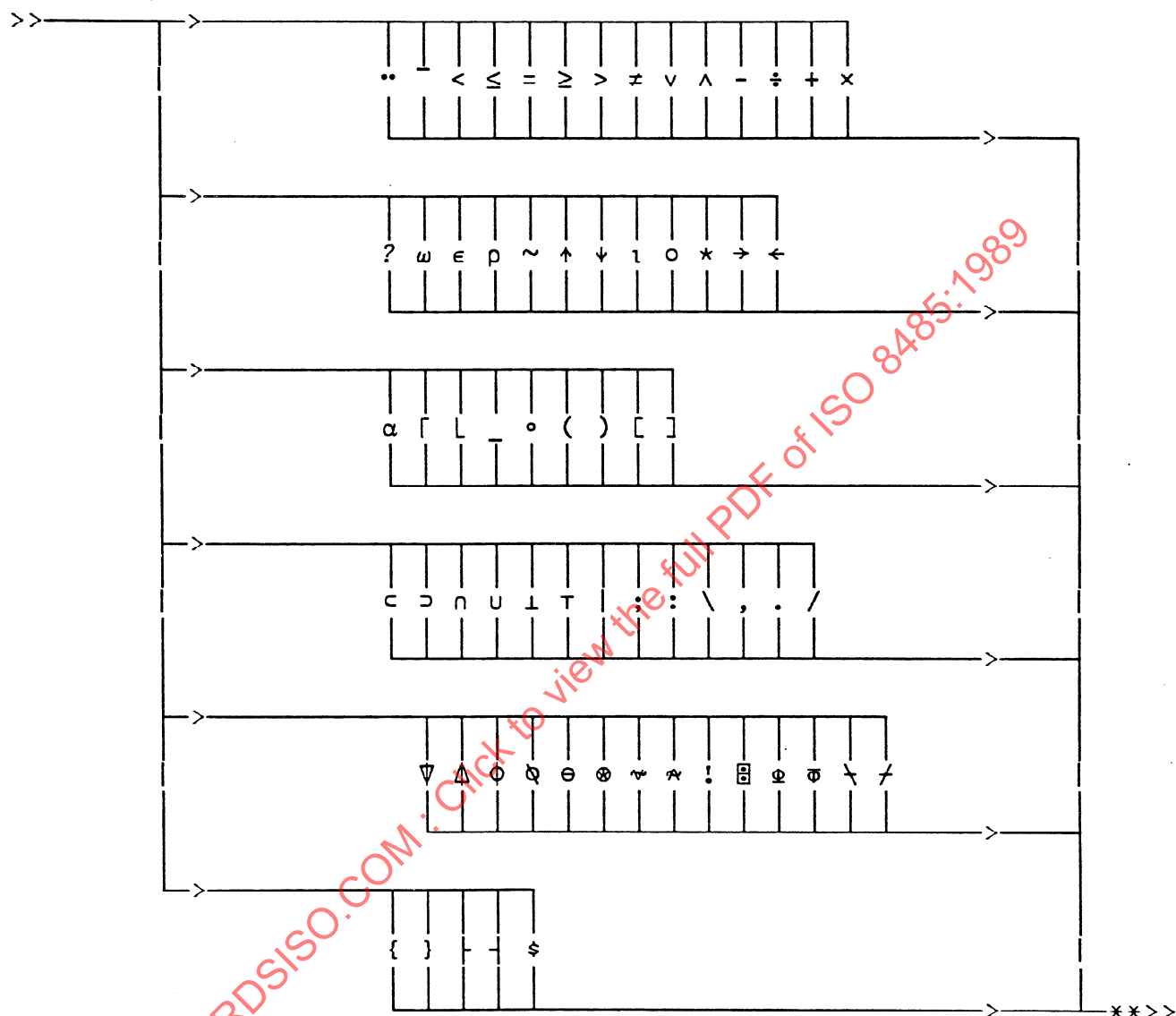
Letter



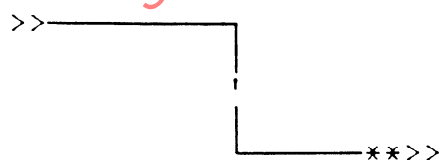
Digit



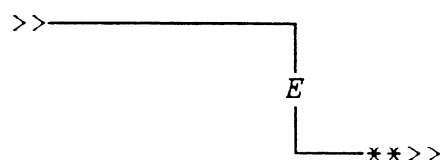
Ideogram



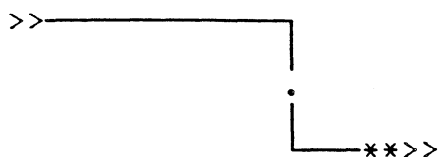
Quote



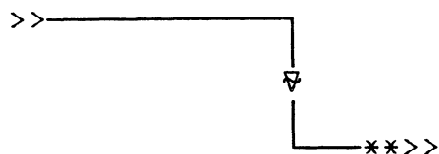
Exponent-Marker



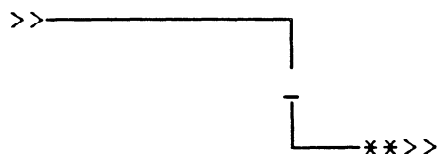
Dot



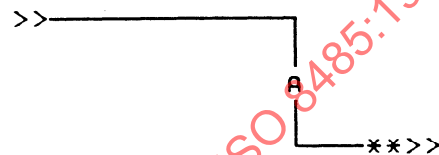
Del-Tilde



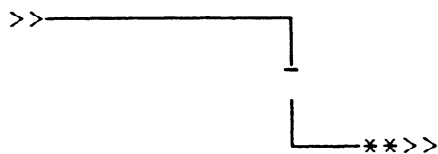
Underbar



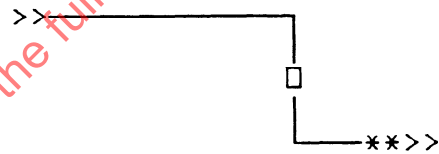
Lamp



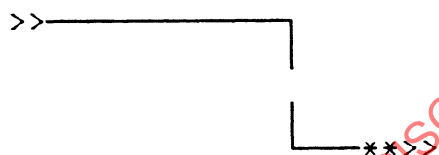
Overbar



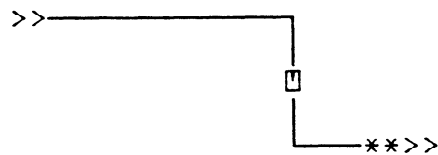
Quad



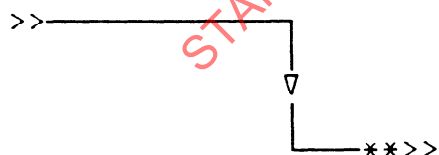
Blank



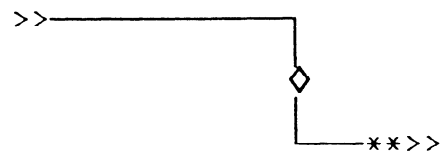
Quote-Quad



Del



Diamond



Example

The example introduced in this subsection is continued through the syntax analysis portion of this standard.

After *evaluate-line* has processed the *current-line*

$ABC \leftarrow FN \ \phi[1+0] \ DEF[1;5 \ 6] \times 3.45E4, \rho 'ABC' \ \text{aCOMMENT}$

current-statement looks like this:

Identification	Content	Class
T01	'ABC'	simple-identifier
T02	'←'	primitive
T03	'FN'	simple-identifier
T04	'ϕ'	distinguished-identifier
T05	'['	primitive
T06	'1'	numeric-literal
T07	'+'	primitive
T08	'0'	numeric-literal
T09	']'	primitive
T10	'DEF'	simple-identifier
T11	'['	primitive
T12	'1'	numeric-literal
T13	':'	primitive
T14	'5 \ 6'	numeric-literal
T15	']'	primitive
T16	'x'	primitive
T17	'3.45E4'	numeric-literal
T18	','	primitive
T19	'ρ'	primitive
T20	'ABC'	character-literal
T21	' '	

Each **token** has a **content**, which is a list of **characters**, and a **class**, which is in the metaclass **lexical-unit**. The **tokens** produced are numbered T01 to T21 for later reference. Note that comments and blanks between tokens are discarded during this tokenisation process.

Note: As the diagram line shows, identifiers and numeric literals are separated by one or more spaces, character-literals, or primitives. 1 ABC'A' is a line, 1ABC'A' is not. No such separation rule applies to primitive function symbols. 1ρ'AB' is a line.

The sequence 3+.4 is parsed numeric-literal, primitive, numeric-literal but the sequence 3+.x is parsed numeric-literal, primitive, primitive, primitive.

A comment may appear at the end of a line or alone on a line.

The **statement-separator-facility** is an **optional-facility**.

Parsing an **identifier token** signals a **limit-error** if the **number-of-items** in the **list** of **characters** is greater than **identifier-length-limit**.

6.1.3 Evaluate-Statement

Evaluate-Statement

Informal Description: Evaluate-statement is performed on **current-statement**, a list of **tokens** found in the **current-context**.

It uses **bind-token-class** to convert **identifiers** to **classified-names** and **constants**.

It uses **literal-conversion** to convert **literals** to **constants**.

It uses the **token-diagram** named **statement** to verify that the statement is properly formed and to resolve certain ambiguous **tokens**. For example, the **token** containing **]** is resolved to either **right-axis-bracket** or **right-index-bracket**.

It calls the evaluation sequence in the subsection **reduce-statement** to convert the **current-statement** to a **result**, which it returns to its caller.

Evaluate-statement is called by **evaluate-line**.

Evaluation Sequence:

For every **index** *I* in the **index-set** of **current-statement**,

Let *T* stand for item *I* of **current-statement**.

If *T* is an **identifier**, set *Q* to **bind-token-class** of *T*.

If *T* is a **literal**, set *Q* to the **literal-conversion** of *T*.

If *T* is a **primitive**, set *Q* to *T*.

If *Q* is an **exception**, return *Q*.

Otherwise, set *T* to *Q*.

Thread **statement** with **current-statement**; if **statement** cannot be matched, signal **syntax-error**.

Rethread **statement** with **current-statement**, taking the following action:

When any **token-diagram** ending in '**—*>>**' is threaded,

Replace the **token** in **current-statement** that matched the diagram with a **token** having the same **content**, but having a **class** given by the name of the **token-diagram**.

Append to **current-statement** as a new first item a **left-end-of-statement token**.

Append to **current-statement** as a new last item a **right-end-of-statement token**.

Set *Z* to **reduce-statement**.

If **mode** is **defined-function** and **current-line-number** is in **current-trace-vector**, set *Z* to **trace-display** of *Z*.

Return *Z*.

Additional Requirement:

This standard permits two orders of evaluation for expressions in brackets, as follows:

The syntax evaluator used here distinguishes axis brackets from index brackets in order to classify a function immediately to the right as monadic or dyadic. For example, the evaluation of $\Phi[pX] + X$ is specified to proceed by first calling monadic plus, then evaluating monadic rho.

The permitted alternative is to evaluate any expression in brackets before determining whether a function immediately to the right is monadic or dyadic. In the example given, this alternative behaviour would evaluate monadic rho before discovering that plus is used monadically.

Note: The distinction can be observed only through side effects.

6.1.4 Bind-Token-Class

Bind-Token-Class of T

Informal Description: **Bind-token-class** is used to bind each **identifier** in **current-statement** to its current **syntactic-unit**; if the **class** of a **token** changes between this point and the time the **token** is used (as it would for example in $F \leftarrow EX'F'$, assuming F were initially a **defined-function**), the change will be detected and reported as a **syntax-error** in the appropriate **phrase-evaluator**.

This prebinding limits the standard to defining the meaning of statements only when the syntax class of their tokens does not change in mid-statement. **Conforming-implementations** may, of course, relax these rules. **Conforming-programs** must abide by them.

Evaluation Sequence:

Let f stand for the **content** of T .

If T is a **simple-identifier**,

If the **current-class** of T is

defined-function, return a **token** of class **defined-function-name** and content f .

niladic-defined-function, return a **token** of class **niladic-defined-function-name** and content f .

nil or **variable**, return a **token** of class **variable-name** and content f .

shared-variable, return a **token** of class **shared-variable-name** and content f .

label, return a **token** of class **constant** and content the **current-content** of T .

Note: Other cases cannot occur.

If T is a **distinguished-identifier**,

If both forms $Z \leftarrow f$ and $Z \leftarrow f \leftarrow B$ occur in the **form-table**, return a **token** of class **system-variable-name** and content f .

If the form $Z \leftarrow f$ occurs in the **form-table**, but the form

$Z \leftarrow f \leftarrow B$ does not, return a **token** of class **niladic-system-function-name** and content f .

If either form $Z \leftarrow f \leftarrow B$ or form $Z \leftarrow A \leftarrow f \leftarrow B$ occurs in the **form-table**, return a **token** of class **system-function-name** with content f .

Otherwise, signal **syntax-error**.

6.1.5 Literal-Conversion

Literal-Conversion of T .

Informal Description: **Literal-conversion** converts T , a token of class **literal**, to a token of class **constant**. The content of such a token is converted from a list of characters to an array.

Evaluation Sequence:

If T is a **character-literal**, generate Z , a **character vector** such that the **ravel-list** of Z is the **content** of T with the initial and final quotes removed, and each pair of adjacent quotes in T replaced by a single quote.

If T is a **numeric-literal**, generate Z , a **numeric vector** such that the **ravel-list** of Z is a list of **numbers**, each of which is obtained by calling the **implementation-algorithm numeric-input-conversion** for the corresponding **numeric-scalar-literal** in the **numeric-literal**.

If the **length** of Z is **greater-than one**, return Z .
Otherwise, return **first-scalar** in Z .

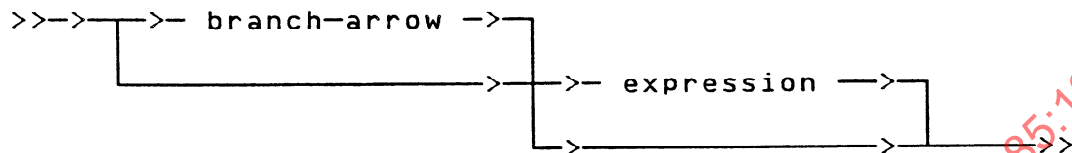
Note: A quote character is represented in a **character-literal** by two adjacent quote characters. A single character between quotes is a scalar. All other cases are **character vector literals**. For example, the **character literal** `' '` is the empty **character vector literal** and the **character literal** `' '' '` is the **character scalar** "quote".

A **numeric literal** containing a single number is a scalar. A **numeric literal** containing two or more numbers is a vector.

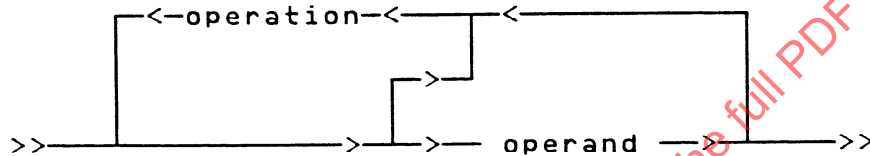
6.1.6 Statement-Analysis Token-Diagrams

Paths containing **ideograms** such as $\otimes$ in these **token-diagrams** match **tokens** whose **class** is **primitive** and whose **content** is the ideogram. Paths containing the word **token**, such as **shared-variable-name token** in **operand**, match **tokens** with the given **class**.

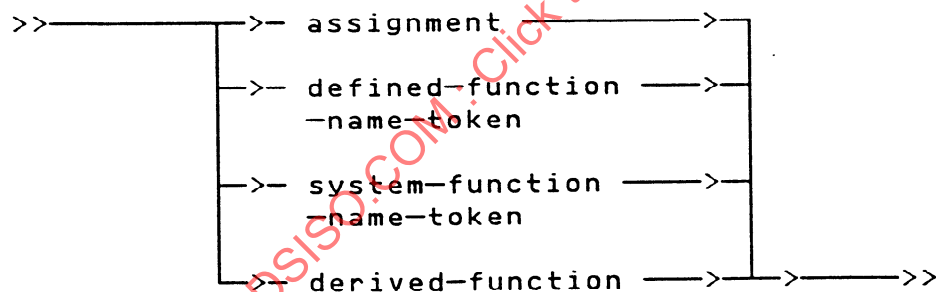
Statement



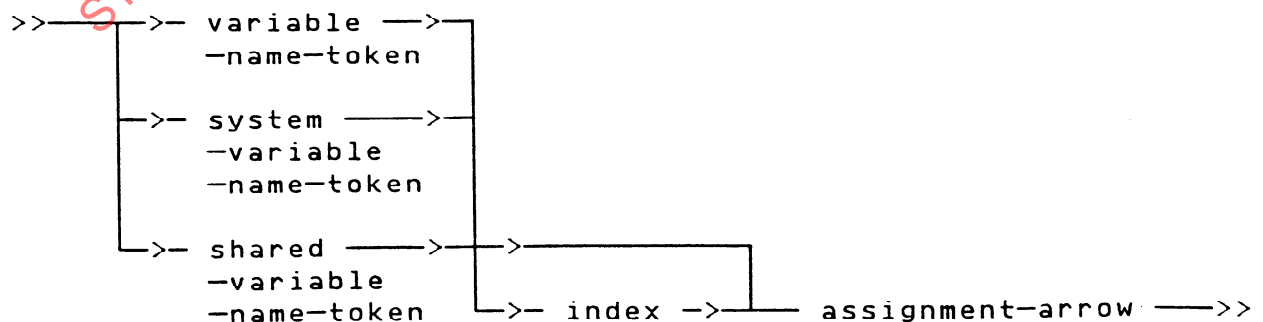
Expression



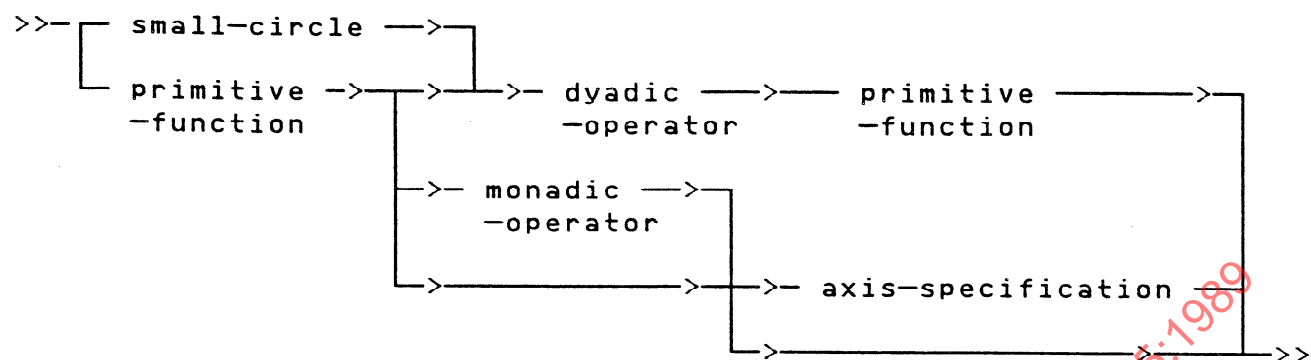
Operation



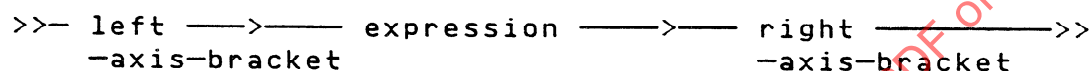
Assignment



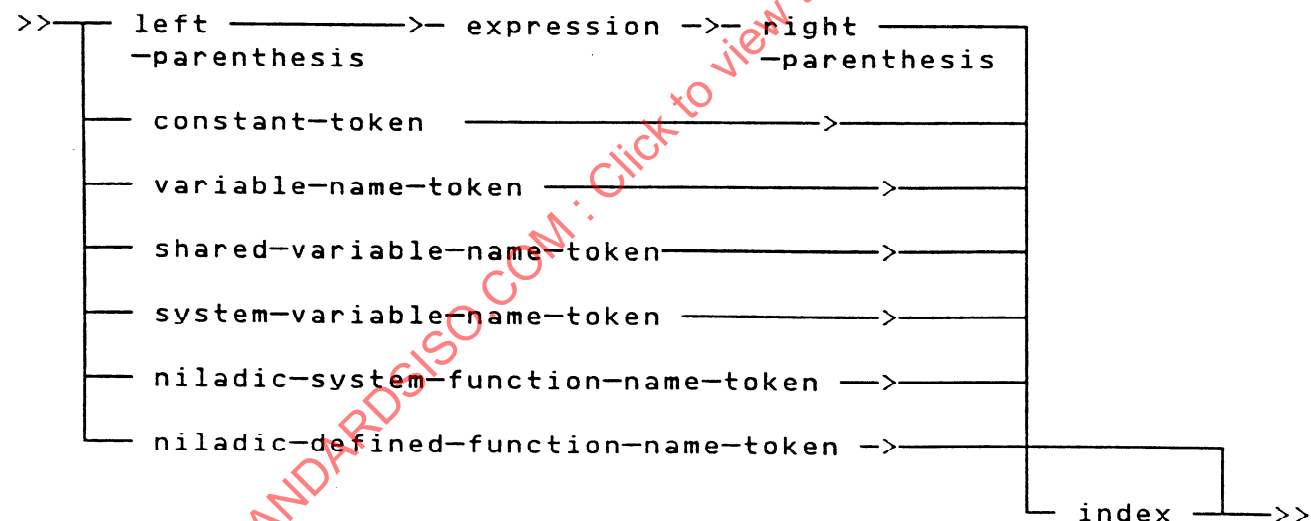
Derived-Function



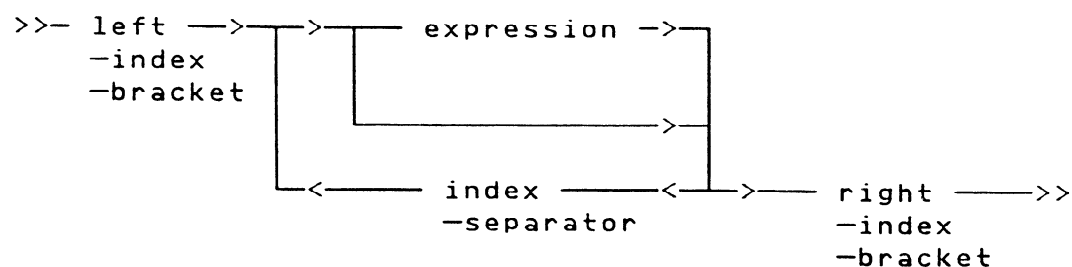
Axis-Specification



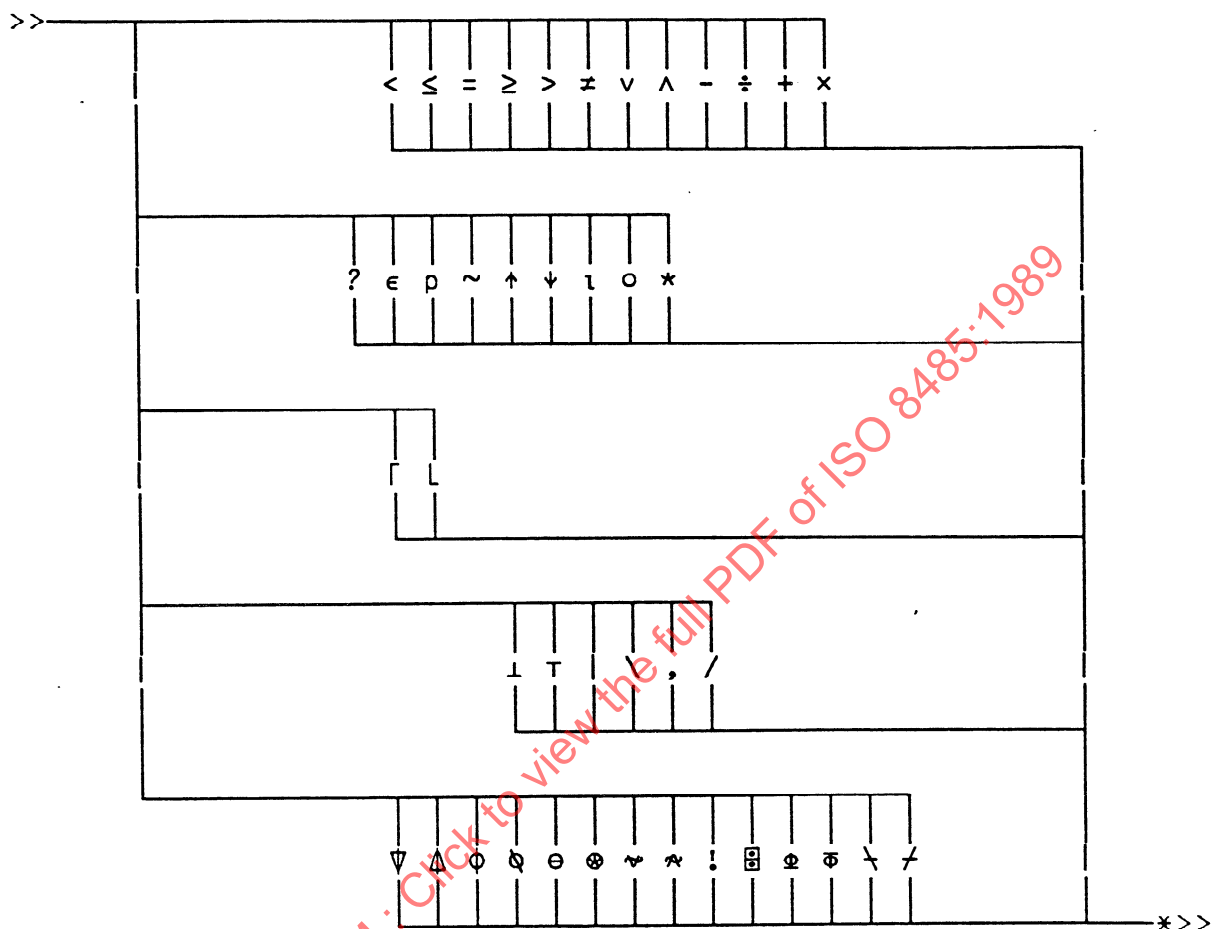
Operand



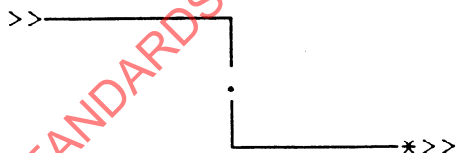
Index



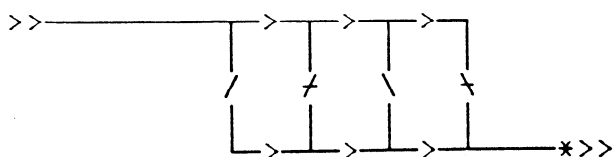
Primitive-Function



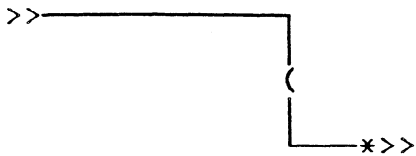
Dyadic-Operator



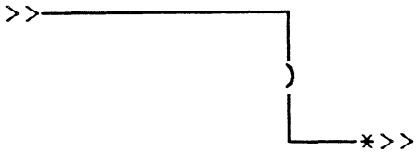
Monadic-Operator



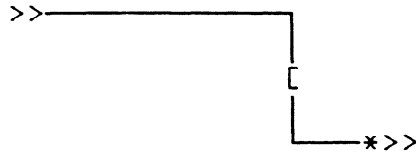
Left-Parenthesis



Right-Parenthesis



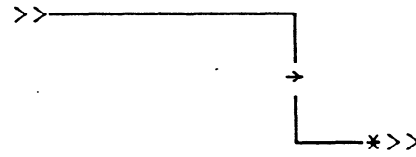
Left-Axis-Bracket



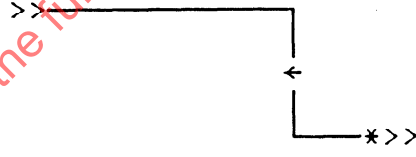
Right-Axis-Bracket



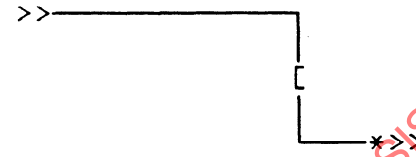
Branch-Arrow



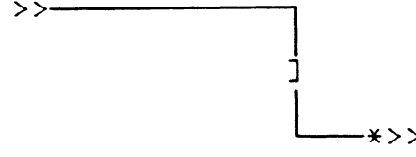
Assignment-Arrow



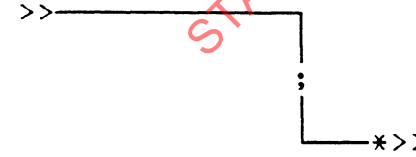
Left-Index-Bracket



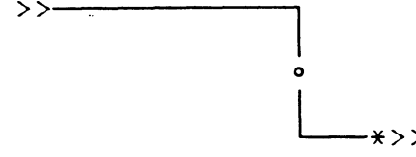
Right-Index-Bracket



Index-Separator



Small-Circle



Example

From the list of characters in current-line,

$ABC \leftarrow FN \ \square \phi [1+0] \ DEF[1;5 \ 6] \times 3.45E4, p'ABC' \ \alpha COMMENT$

evaluate-line generated a list of tokens and stored it in current-statement. Here, evaluate-statement has replaced that list with a new list of tokens.

Identification	Content	Class
T00		left-end-of-statement
T01	'ABC'	variable-name
T02		assignment-arrow
T03	'FN'	defined-function-name
T04	'□'	system-variable-name
T05	'φ'	primitive-function
T06		left-axis-bracket
T07	1	constant
T08	'+'	primitive-function
T09	0	constant
T10		right-axis-bracket
T11	'DEF'	variable-name
T12		left-index-bracket
T13	1	constant
T14		index-separator
T15	5 6	constant
T16		right-index-bracket
T17	'×	primitive-function
T18	34500	constant
T19	','	primitive-function
T20	'p'	primitive-function
T21	'ABC'	constant
T22		right-end-of-statement

The new list, shown above, begins with a left-end-of-statement token and ends with a right-end-of-statement token. Old identifier tokens have a new class and the same content; old literal tokens are now constants whose content is an appropriate array; old primitive tokens are now either primitive-functions whose content is the character identifying the primitive function, or are grouping signs such as right-axis-bracket with no specified content.

6.2 REDUCE-STATEMENT

Reduce-Statement

Informal Description: Reduce-statement converts **current-statement**, a list of **syntactic-units**, to a **result** by decomposing it into shorter lists of **syntactic-units** called **phrases**, then calling procedures termed **phrase-evaluators** to convert these phrases into **tokens**.

The letters Z , K , and J in this evaluation sequence refer to the graphic symbols found in the **resultant-prefix** column of Table 3.

Evaluation Sequence:

Set **current-stack** to the **empty-list** of **tokens**.

Repeat:

Find the first entry in the **phrase-table** whose **pattern** matches a **prefix** of **current-stack**.
If there is no matching entry,

If **current-statement** is empty, signal **syntax-error**.
Otherwise,

Remove the last **token** from **current-statement**.
Append it to **current-stack** as a new first item.

If there is a matching entry,

Let **p** stand for the **prefix** of the **current-stack** that matched the entry.
Let **r** stand for the **resultant-prefix** of the entry.
Let **s** stand for the **phrase-evaluator** of the entry.

Call **s** and set **y** to the **token** it returns.

If **s** is **process-end-of-statement**, return **y**.

If **y** is an **exception**, return **y**.

Otherwise, replace **p** with **r** in which **y** has been substituted for Z , K or J according to whether **y** is a **result**, a **complete-index-list** or a **partial-index-list**.

(End of repeated block)

Table 3 — The Phrase Table

Pattern	Phrase Evaluator	Resultant-Prefix
(B)	Remove-Parentheses	Z
N	Evaluate-Niladic-Function	Z
X F B	Evaluate-Monadic-Function	X Z
X F [C] B	Evaluate-Monadic-Function	X Z
X F M B	Evaluate-Monadic-Operator	X Z
X F M [C] B	Evaluate-Monadic-Operator	X Z
A F B	Evaluate-Dyadic-Function	Z
A F [C] B	Evaluate-Dyadic-Function	Z
A F D G B	Evaluate-Dyadic-Operator	Z
A ° D G B	Evaluate-Dyadic-Operator	Z
A [K]	Evaluate-Indexed-Reference	Z
V [K] ← B	Evaluate-Indexed-Assignment	Z
V ← B	Evaluate-Assignment	Z
V	Evaluate-Variable	Z
]	Build-Index-List	J]
; I	Build-Index-List	J
; B I	Build-Index-List	J
[I	Build-Index-List	[K
[B I	Build-Index-List	[K
L R	Process-End-of-Statement	-
L B R	Process-End-of-Statement	.
L → B R	Process-End-of-Statement	.
L → R	Process-End-of-Statement	.

Legend:

A, B, Z match **result**.
 D matches **dyadic-operator**.
 F, G match **defined-function-name**,
primitive-function, or
system-function-name.
 I, J match **partial-index-list**.
 C, K match **complete-index-list**.
 L matches **left-end-of-statement**.
 M matches **monadic-operator**.
 N matches **niladic-defined-function-name**
 or **niladic-system-function-name**.
 R matches **right-end-of-statement**.
 V matches **variable-name**,
system-variable-name, or
shared-variable-name.

X matches **assignment-arrow**,
branch-arrow, **defined-function-name**,
index-separator, **left-axis-bracket**,
left-end-of-statement, **left-index-bracket**,
left-parenthesis, **primitive-function**,
system-function-name, or
right-axis-bracket.
 (matches **left-parenthesis**.
) matches **right-parenthesis**.
 [matches **left-axis-bracket** or
left-index-bracket.
] matches **right-axis-bracket** or
right-index-bracket.
 ° matches **small-circle**.
 ; matches **index-separator**.
 ← matches **assignment-arrow**.
 → matches **branch-arrow**.

The graphic symbols in the **Pattern** and **Resultant-Prefix** columns of Table 3 designate lists of **syntactic-units**. Each graphic symbol matches **tokens** of the designated **classes** or **metaclasses**.

Note: The graphic symbols employed are chosen to be suggestive of the list of **characters** that give rise to such phrases.

The **build-index-list** entries are called with *B* bound to a value and *I* bound to an **partial-index-list**; they return either another **partial-index-list** (*J*) or a **complete-index-list** (*K*). Brackets are not passed as arguments or returned by **build-index-list**; they are preserved by **reduce-statement** to make the patterns more obvious.

Example

This example is continued from **evaluate-statement**.

The line being evaluated is

ABC	←	FN	□	Φ	[	1	+	0	]	DEF	[	1	;	5	6	]	×	34500	,	p	'ABC'
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	

T00 is left-end-of-statement, and T22 is right-end-of-statement.

The tokens T00 through T22 initially form the columns of Figure 1. The rows of the figure show the actions taken because of the pattern or lack of pattern in **current-stack**.

Consider step 5: the line fragment , p 'ABC' matches the phrase X F B as follows: X matches token T19 because , is a **primitive-function**. F matches token T20 because p is a **primitive-function**. B matches token T21 because 'ABC' is a **constant**. Token T22, right-end-of-statement, is not considered because the pattern X F B is concerned with only the first three tokens in **current-stack**.

The **phrase-evaluator** associated with X F B is **evaluate-monadic-function**. This **phrase-evaluator**, seeing that p is a **primitive function**, searches the **form-table** for $Z \leftarrow p \ B$, and calls the corresponding evaluation sequence, **shape**. **Shape** returns a **constant**, the one-element-vector containing three.

This becomes Z in the **resultant-prefix** column for X F B. **Current-stack** now holds three tokens: (**primitive-function**; ,), (**constant**; , 3) and (**right-end-of-statement**;). Since no entry in the **phrase-table** has an entry whose **pattern** matches a **prefix** of **current-stack**, the token T18 (**constant**; 34500) is added to **current-stack**. A **prefix** of **current-stack** now matches a **pattern** (A F B matches (**constant**; 34500), (**primitive-function**; ,), (**constant**; , 3)), so the corresponding **phrase-evaluator** (**evaluate-dyadic-function**) is called.

```

1 GET NEXT TOKEN T22                ( )
2 GET NEXT TOKEN T21                (R)
3 GET NEXT TOKEN T20                (B R)
4 GET NEXT TOKEN T19                (F B R)
5 EVALUATE MONADIC FUNCTION          (<X F B>R)
6 GET NEXT TOKEN T18                (<X Z> R)
7 EVALUATE DYADIC FUNCTION           (<A F B> R)
8 GET NEXT TOKEN T17                (Z R)
9 GET NEXT TOKEN T16                (F B R)
10 BUILD INDEX LIST                  (<]>F B R)
11 GET NEXT TOKEN T15                (J ] F B R)
12 GET NEXT TOKEN T14                (B I ] F B R)
13 BUILD INDEX LIST                  (<; B I>] F B R)
14 GET NEXT TOKEN T13                (J ] F B R)
15 GET NEXT TOKEN T12                (B I ] F B R)
16 BUILD INDEX LIST                  (<[ B I> ] F B R)
17 GET NEXT TOKEN T11                ([ K ] F B R)
18 EVALUATE VARIABLE                  (<V>[ K ] F B R)
19 EVALUATE INDEXED REFERENCE        (<A [ K ]>F B R)
20 EVALUATE DYADIC FUNCTION          (<A F B> R)
21 GET NEXT TOKEN T10                (Z R)
22 BUILD INDEX LIST                  (<]>B R)
23 GET NEXT TOKEN T09                (I ] B R)
24 GET NEXT TOKEN T08                (B I ] B R)
25 GET NEXT TOKEN T07                (F B I ] B R)
26 EVALUATE DYADIC (<A F B>I ] B FUNCTION R)
27 GET NEXT (Z I ] B TOKEN T06 R)
28 BUILD INDEX (<[ B I>] B LIST R)
29 GET NEXT ([ C ] B TOKEN T05 R)
30 GET NEXT (F [ C ] B TOKEN T04 R)
31 EVALUATE (<V>F [ C ] B VARIABLE R)
32 EVALUATE (<A F [ C ] B> DYADIC FUNCTION R)
33 GET NEXT (Z TOKEN T03 R)
34 (F B GET NEXT TOKEN T02 R)
35 (<X F B> EVALUATE MONADIC FUNCTION R)
36 (X Z GET NEXT TOKEN T01 R)
37 (<V * B> EVALUATE ASSIGNMENT R)
38 (Z GET NEXT TOKEN T00 R)
39 * (<L B END OF STATEMENT R>)

```

Figure 1 — Statement Evaluation

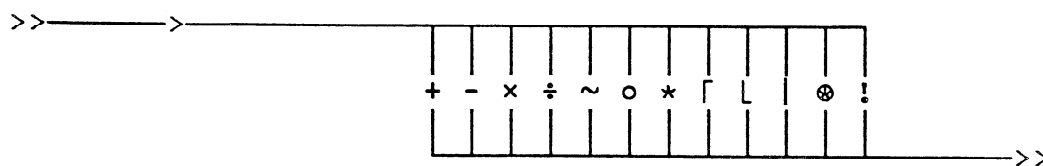
Extent of **current-stack** shown by (and). **Prefix** (when matched) shown by < and >.

6.3 THE PHRASE EVALUATORS

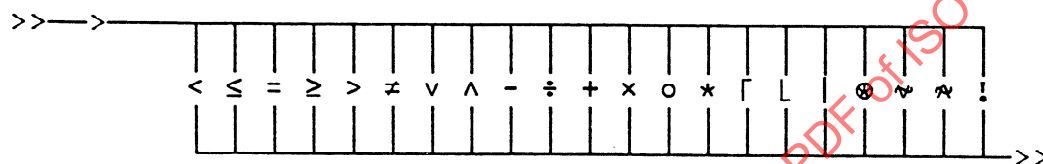
Informal Description: Each **phrase-evaluator** takes a **phrase** and reduces it to a single **token**. The **form-table** used by the **phrase-evaluators** is given as Table 4.

6.3.1 Diagrams

Primitive-Monadic-Scalar-Function



Primitive-Dyadic-Scalar-Function



STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

6.3.2 Remove-Parentheses

Pattern (*B*)

Evaluation Sequence:

If *B* is **nil**, signal **value-error**.
 If *B* is a **branch**, signal **value-error**.
 Otherwise, return *B*.

Note: (Φ ' ') and (Φ ' $\rightarrow$ 3 ') fail with a **value-error**. ($A \leftarrow$ 3) and (Φ ' $A \leftarrow$ 3 ') do not display, since the **token** returned by **remove-parentheses** is a **committed-value**, not a **constant**; for the same reason, ($\square \leftarrow$ 3) displays the value **three** only once. ($\rightarrow$ 3) fails with a **syntax-error** in **evaluate-statement**.

($\rightarrow$) also fails in **evaluate-statement**, but (Φ ' $\rightarrow$ ') succeeds; **evaluate-statement** threads line successfully, and **remove-parentheses** receives and returns an **escape token**.

6.3.3 Evaluate-Niladic-Function

Pattern *N*

Evaluation Sequence:

Let *n* stand for the **content** of *N*.
 If *N* is a **niladic-defined-function-name**,

If the **current-class** of *N* is **niladic-defined-function**,

Search the **form-table** for $Z \leftarrow DFN$.
 Call the corresponding evaluation sequence, passing *n* as the value of *DFN*.
 Return the **token** it returns.

Otherwise, signal **syntax-error**.

If *N* is a **niladic-system-function-name**,

Search the **form-table** for $Z \leftarrow n$.
 If it is not found, signal **syntax-error**.
 Otherwise, call the corresponding evaluation sequence.
 Return the **token** it returns.

Note: This phrase evaluator checks for a change in syntax class.

6.3.4 Evaluate-Monadic-Function

Pattern $X \ F \ B$

Pattern $X \ F[C] \ B$

Evaluation Sequence:

If B is not a **value**, signal **value-error**.

Let f stand for the **content** of F .

For pattern $X \ F \ B$

If F is a **defined-function-name**,

If the **current-class** of F is **defined-function**,

Search the **form-table** for $Z \leftarrow DFN \ B$.

Call the corresponding evaluation sequence, passing f as the value of DFN .

Return the **token** it returns.

Otherwise, signal **syntax-error**.

If F is a **primitive-function** or a **system-function-name**,

If F matches **primitive-monadic-scalar-function** and B is not a **scalar**, perform **monadic-scalar-extension** as follows:

Return a **numeric array** Z such that the **shape-list** of Z is the **shape-list** of B and for all I in the **index-set** of the **ravel-list** of Z , item I of the **ravel-list** of Z is f (item I of the **ravel-list** of B).

Otherwise,

Search the **form-table** for $Z \leftarrow f \ B$.

If it is not found, signal **syntax-error**.

Otherwise, call the corresponding evaluation sequence.

Return the **token** it returns.

For pattern $X \ F[C] \ B$

If F is not a **primitive-function**, signal **syntax-error**.

If **index-origin** is **nil**, signal **implicit-error**.

Let $C1$ stand for the **first-item** in the **index-list** C .

If $C1$ is not a **scalar** or **one-element-vector**, signal **axis-error**.

If any **item** of the **ravel-list** of $C1$ is not a **number**, signal **axis-error**.

Set K to $C1$ **plus** (**one minus index-origin**).

Search the **form-table** for $Z \leftarrow f[K] \ B$.

If it is not found, signal **syntax-error**.

Otherwise, call the corresponding evaluation sequence and return the **token** it returns.

Additional Requirements:

The order in which the individual items of Z are produced during **monadic-scalar-extension** is not specified by this standard.

If any call of f signals an error during **monadic-scalar-extension**, that error is returned as the result of **evaluate-monadic-function**.

Note: This phrase evaluator checks for a change in syntax class.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

6.3.5 Evaluate-Monadic-Operator

Pattern $X F M B$

Pattern $X F M[C] B$

Evaluation Sequence:

If B is not a **value**, signal **value-error**.

Let m stand for the **content** of M .

Let f stand for the **content** of F .

For pattern $X F M B$

If F is not a **primitive-function**, signal **syntax-error**.
Otherwise,

Search the **form-table** for $Z \leftarrow f m B$.

If it is not found, signal **syntax-error**.

Otherwise, call the corresponding evaluation sequence, passing **token** F as the value of f .
Return the **token** it returns.

For pattern $X F M[C] B$

If F is not a **primitive-function**, signal **syntax-error**.

If **index-origin** is **nil**, signal **implicit-error**.

Let $C1$ stand for the **first-item** in the **index-list** C .

If $C1$ is not a **scalar** or **one-element-vector**, signal **axis-error**.

If any **item** of the **ravel-list** of $C1$ is not a **number**, signal **axis-error**.

Set K to $C1$ **plus** (**one minus index-origin**).

Search the **form-table** for $Z \leftarrow f m[K] B$.

If it is not found, signal **syntax-error**.

Otherwise, call the corresponding evaluation sequence, passing **token** F as the value of f .
Return the **token** it returns.

6.3.6 Evaluate-Dyadic-Function

Pattern $A \ F \ B$

Pattern $A \ F[C] \ B$

Evaluation Sequence:

If A is not a **value**, signal **value-error**.

If B is not a **value**, signal **value-error**.

Let f stand for the **content** of F .

For pattern $A \ F \ B$

If F is a **defined-function-name**,

If the **current-class** of F is **defined-function**,

Search the **form-table** for $Z \leftarrow A \ DFN \ B$.

Call the corresponding evaluation sequence, passing f as the value of DFN .

Return the **token** it returns.

Otherwise, signal **syntax-error**.

If F is a **primitive-function** or a **system-function-name**,

If F matches **primitive-dyadic-scalar-function** and A and B are not both **scalars**, perform **dyadic-scalar-extension** as follows:

If the **rank** of A differs from the **rank** of B ,

If A is a **scalar** or **one-element-vector** and B is not a **scalar**, return $((\rho B) \rho A) f \ B$.

If B is a **scalar** or **one-element-vector**, return $A \ f \ (\rho A) \rho B$.

Otherwise, signal **rank-error**.

If the **shape-list** of A differs from the **shape-list** of B , signal **length-error**.

Return Z , an array such that the **shape-list** of Z is the same as the **shape-list** of A , the **type** of Z is **numeric**, and the **ravel-list** of Z is such that, for all I in the **index-set** of the **ravel-list** of Z , item I of the **ravel-list** of Z is (item I of the **ravel-list** of A) f (item I of the **ravel-list** of B).

Otherwise, search the **form-table** for $Z \leftarrow A \ f \ B$.

If it is not found, signal **syntax-error**.

Call the corresponding evaluation sequence and return the **token** it returns.

For pattern $A \ F[C] \ B$

If F is not a **primitive-function**, signal **syntax-error**.

If **index-origin** is **nil**, signal **implicit-error**.

Let $C1$ stand for the **first-item** in the **index-list** C .

If $C1$ is not a **scalar** or **one-element-vector**, signal **axis-error**.

If any **item** of the **ravel-list** of $C1$ is not a **number**, signal **axis-error**.

Set K to $C1$ **plus** (one minus **index-origin**).

Search the **form-table** for $Z \leftarrow A \ f[K] \ B$.

If it is not found, signal **syntax-error**.

Otherwise, call the corresponding evaluation sequence, passing **token** F as the value of f .

Return the **token** it returns.

Additional Requirements:

There is an intentional forward reference to **shape** and **reshape** in the description of **dyadic-scalar-extension**.

The order in which the individual items of Z are produced during **dyadic-scalar-extension** is not specified by this standard.

If any call of **f** signals an error during **dyadic-scalar-extension**, that error is returned as the result of **evaluate-dyadic-function**.

Note: **Dyadic-scalar-extension** is intentionally stricter than it is in existing systems. For example, $(1 \ 1 \ 1) + 1 \ 3$ signals a **rank-error** and $1 \ 2 \ + \ ,1$ signals a **length-error**.

This phrase evaluator checks for a change in syntax class.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

6.3.7 Evaluate-Dyadic-Operator

Pattern $A \ F \ D \ G \ B$

Pattern $A \circ \ D \ G \ B$

Evaluation Sequence:

If A is not a **value**, signal **value-error**.

If B is not a **value**, signal **value-error**.

Let d stand for the **content** of D .

For pattern $A \ F \ D \ G \ B$

If F or G is not a **primitive-function**, signal **syntax-error**.
Otherwise,

Search the **form-table** for $Z \leftarrow A \ f \ d \ g \ B$.

If it is not found, signal **syntax-error**.

If it is found, call the corresponding evaluation sequence, passing **token f** as the value of F and **token g** as the value of G .

Return the **token** it returns.

For pattern $A \circ \ D \ G \ B$

If G is not a **primitive-function**, signal **syntax-error**.
Otherwise,

Search the **form-table** for $Z \leftarrow A \circ \ d \ g \ B$.

If it is not found, signal **syntax-error**.

If it is found, call the corresponding evaluation sequence, passing **token g** as the value of G .

Return the **token** it returns.

6.3.8 Evaluate-Indexed-Reference

Pattern $A[K]$

Evaluation Sequence:

If A is not a **value**, signal **value-error**.
If **index-origin** is **nil**, signal **implicit-error**.

If the **number-of-items** in the **index-list** K differs from the **rank** of A , signal **rank-error**.

If the **rank** of A is **greater-than one**,

Search the **form-table** for $Z \leftarrow A[I]$.

Call the corresponding evaluation sequence, passing K as the value of I .
Return the **token** it returns.

Otherwise,

If **first-item** in K is an **elided-index-marker**, return A .
Otherwise,

Set X to **first-item** in the **index-list** K .

If any **item** of the **ravel-list** of X is not a **near-integer**, signal **domain-error**.

Generate $X1$, a **numeric array** with the **shape-list** of X such that each **item** of the **ravel-list** of $X1$ is **(one minus index-origin) plus the integer-nearest-to** X .

If any **item** of the **ravel-list** of $X1$ is not in the **index-set** of A , signal **index-error**.

Return Z , an array with the **type** of A and the **shape-list** of $X1$, such that for each **integer** I in the **index-set** of Z , **item** I of the **ravel-list** of Z is **item** J of the **ravel-list** of A , where J is **item** I of the **ravel-list** of $X1$.

Note: Since an **index-list** will never have zero items, indexing will always signal a **rank-error** when argument A is a **scalar**.

6.3.9 Evaluate-Assignment

Pattern $V \leftarrow B$

Evaluation Sequence:

- If B is not a **value**, signal **value-error**.
- If V is a **shared-variable-name**,

If the **current-class** of V is **shared-variable**,

Search the **form-table** for $Z \leftarrow SHV \leftarrow B$.

Call the corresponding evaluation sequence, passing **token** V as the value of SHV .
Return the **token** it returns.

Otherwise, signal **syntax-error**.

- If V is a **system-variable-name**,

Search the **form-table** for $Z \leftarrow q \leftarrow B$, where q is the **content** of V .

If it is not found, signal **syntax-error**.

Otherwise,

Call the corresponding evaluation sequence

Return the **token** it returns

- If V is a **variable-name**,

If the **current-class** of V is **nil** or **variable**,

Set the **current-referent** of V to a **token** whose **class** is **variable** and whose **content** is the **content** of B .

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Otherwise, signal **syntax-error**.

*Note: The phrase $ABC \leftarrow \emptyset \rightarrow 3$ yields **value-error**.*

*The phrase $V \leftarrow \square SVR \quad ! V$ where V was a **shared-variable** yields **syntax-error**.*

6.3.10 Evaluate-Indexed-Assignment

Pattern $V[K] \leftarrow B$

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.

If B is not a **value**, signal **value-error**.

If V is a **shared-variable-name**,

If the **current-class** of V is **shared-variable**,

Search the **form-table** for $Z \leftarrow SHV[I] \leftarrow B$.

Call the corresponding evaluation sequence, passing **token** V as the value of SHV , and K as the value of I .

Return the **token** it returns.

Otherwise, signal **syntax-error**.

If V is a **system-variable-name**,

Search the **form-table** for $Z \leftarrow q[I] \leftarrow B$, where q is the **content** of V .

If it is not found, signal **syntax-error**.

Otherwise,

Call the corresponding evaluation sequence, passing K as the value of I .

Return the **token** it returns.

If V is a **variable-name**,

If the **current-class** of V is **nil**, signal **value-error**.

If the **current-class** of V is a **variable**,

Search the **form-table** for $Z \leftarrow V[I] \leftarrow B$.

Call the corresponding evaluation sequence, passing K as the value of I .

Return the **token** it returns.

Otherwise, signal **syntax-error**.

6.3.11 Evaluate-Variable

Pattern V

Evaluation Sequence:

If V is a **shared-variable-name**,

If the **current-class** of V is **shared-variable**,

Search the **form-table** for $Z \leftarrow SHV$.

Call the corresponding evaluation sequence, passing **token** V as the value of SHV .

Return the **token** it returns.

Otherwise, signal **syntax-error**.

If V is a **system-variable-name**,

Search the **form-table** for $Z \leftarrow q$, where q is the **content** of V .

Call the corresponding evaluation sequence.

Return the **token** it returns.

If V is a **variable-name**,

If the **current-class** of V is **nil**, signal **value-error**.

If the **current-class** of V is **variable**, return the **current-content** of V .

Otherwise, signal **syntax-error**.

6.3.12 Build-Index-List

Pattern]

Pattern ; *I*

Pattern ; *B I*

Pattern [*I*

Pattern [*B I*

Evaluation Sequence:

For pattern]

Return *J*, a **partial-index-list** with **content** the index-list of length **zero**.

For pattern ; *I*

Return *J*, a **partial-index-list** with **content** *Z*, an **index-list** such that the **first-item** in *Z* is an **elided-index-marker** and the **rest-of** *Z* is *I*.

For pattern ; *B I*

Return *J*, a **partial-index-list** with **content** *Z*, an **index-list** such that the **first-item** in *Z* is *B* and the **rest-of** *Z* is *I*.

For pattern [*I*

Return *J*, a **complete-index-list** with **content** *Z*, an **index-list** such that the **first-item** in *Z* is an **elided-index-marker** and the **rest-of** *Z* is *I*.

For pattern [*B I*

Return *J*, a **complete-index-list** with **content** *Z*, an **index-list** such that the **first-item** in *Z* is *B* and the **rest-of** *Z* is *I*.

6.3.13 Process-End-of-Statement

Pattern $L R$

Pattern $L A R$

Pattern $L \rightarrow R$

Pattern $L \rightarrow A R$

Evaluation Sequence:

For pattern $L R$

Return a **token** whose **class** is **nil**.

For pattern $L A R$

Return A .

For pattern $L \rightarrow R$

Return a **token** whose **class** is **escape**.

For pattern $L \rightarrow A R$

If the **rank** of A is **greater-than one**, signal **rank-error**.

If A is **empty**, return a **token** whose **class** is **nil**.

Otherwise, set $A1$ to the **first-scalar** in A .

If $A1$ is not a **near-integer**, signal **domain-error**.

Return a **token** whose **class** is **branch** and whose **content** is the **numeric-scalar** with value the **integer-nearest-to** $A1$.

6.4 THE FORM TABLE

The **form-table** is the list of all **lists** of **syntactic-units** for which evaluation sequences exist.

The following matching rules apply in the **form-table**.

A, B, Z match **constant**.

I, K match **complete-index-list**.

f, g match **primitive-function**.

A given ideogram, such as $\otimes$, matches a **primitive-function token** that contains it.

A given **distinguished-identifier**, such as $\square IO$, matches any **system-variable-name token** or **system-function-name token** that contains it.

The behaviour of operations in the **form-table** that do not create new **contexts** is **atomic**.

This behaviour is observable only for those operations that have **side-effects**. For example, if any of the elements of an argument array is not in the domain of **roll**, the value of the system parameter **random-link** following execution will be as it was when **roll** was called.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

Table 4 — The Form Table

Form	Operation Name	Page
$Z \leftarrow + B$	Conjugate	84
$Z \leftarrow - B$	Negative	84
$Z \leftarrow \times B$	Signum	85
$Z \leftarrow \div B$	Reciprocal	85
$Z \leftarrow \lfloor B$	Floor	86
$Z \leftarrow \lceil B$	Ceiling	86
$Z \leftarrow * B$	Exponential	87
$Z \leftarrow \otimes B$	Natural Logarithm	87
$Z \leftarrow B$	Magnitude	88
$Z \leftarrow ! B$	Factorial	88
$Z \leftarrow \circ B$	Pi times	89
$Z \leftarrow \sim B$	Not	89
$Z \leftarrow A + B$	Plus	91
$Z \leftarrow A - B$	Minus	91
$Z \leftarrow A \times B$	Times	92
$Z \leftarrow A \div B$	Divide	92
$Z \leftarrow A \lceil B$	Maximum	93
$Z \leftarrow A \lfloor B$	Minimum	93
$Z \leftarrow A * B$	Power	94
$Z \leftarrow A \otimes B$	Logarithm	94
$Z \leftarrow A B$	Residue	95
$Z \leftarrow A ! B$	Binomial	96
$Z \leftarrow A \circ B$	Circular Functions	97
$Z \leftarrow A \wedge B$	And	99
$Z \leftarrow A \vee B$	Or	99
$Z \leftarrow A \neg B$	Nand	100
$Z \leftarrow A \sim B$	Nor	100
$Z \leftarrow A = B$	Equal	101
$Z \leftarrow A < B$	Less than	103
$Z \leftarrow A \leq B$	Less than or equal to	103
$Z \leftarrow A \neq B$	Not equal	104
$Z \leftarrow A \geq B$	Greater than or equal to	104
$Z \leftarrow A > B$	Greater than	105
$Z \leftarrow , B$	Ravel	107
$Z \leftarrow \rho B$	Shape	108
$Z \leftarrow 1 B$	Index Generator	109
$Z \leftarrow A \rho B$	Reshape	110
$Z \leftarrow A , B$	Join	111
$Z \leftarrow f / B$	Reduction	114
$Z \leftarrow f / [K] B$	Reduction	114
$Z \leftarrow f \neq B$	Reduction	114
$Z \leftarrow f \neq [K] B$	Reduction	114
$Z \leftarrow f \setminus B$	Scan	116
$Z \leftarrow f \setminus [K] B$	Scan	116
$Z \leftarrow f \setminus B$	Scan	116
$Z \leftarrow f \setminus [K] B$	Scan	116
$Z \leftarrow A \circ . f B$	Outer Product	117
$Z \leftarrow A f . g B$	Inner Product	118
$Z \leftarrow ? B$	Roll	119
$Z \leftarrow \Delta B$	Grade Up	121
$Z \leftarrow \nabla B$	Grade Down	122
$Z \leftarrow \Phi B$	Reverse	123

Table 4 — The Form Table (continued)

Form	Operation Name	Page
$Z \leftarrow \ominus B$	Reverse	123
$Z \leftarrow \Phi[K] B$	Reverse	123
$Z \leftarrow \Theta[K] B$	Reverse	123
$Z \leftarrow \boxtimes B$	Monadic Transpose	124
$Z \leftarrow \boxminus B$	Matrix Inverse	125
$Z \leftarrow \Phi B$	Execute	126
$Z \leftarrow A , [K] B$	Join Along an Axis	127
$Z \leftarrow A \downarrow B$	Index of	130
$Z \leftarrow A \in B$	Member of	131
$Z \leftarrow A ? B$	Deal	132
$Z \leftarrow A / B$	Compress	133
$Z \leftarrow A \neq B$	Compress	133
$Z \leftarrow A / [K] B$	Compress	133
$Z \leftarrow A \neq [K] B$	Compress	133
$Z \leftarrow A \setminus B$	Expand	135
$Z \leftarrow A \setminus B$	Expand	135
$Z \leftarrow A \setminus [K] B$	Expand	135
$Z \leftarrow A \setminus [K] B$	Expand	135
$Z \leftarrow A \Phi B$	Rotate	137
$Z \leftarrow A \Theta B$	Rotate	137
$Z \leftarrow A \Phi [K] B$	Rotate	137
$Z \leftarrow A \Theta [K] B$	Rotate	137
$Z \leftarrow A \perp B$	Base Value	139
$Z \leftarrow A \top B$	Representation	140
$Z \leftarrow A \boxtimes B$	Dyadic Transpose	142
$Z \leftarrow A \uparrow B$	Take	144
$Z \leftarrow A \downarrow B$	Drop	145
$Z \leftarrow A \boxdiv B$	Matrix Divide	146
$Z \leftarrow A[I]$	Indexed Reference	147
$Z \leftarrow V[I] \leftarrow B$	Indexed Assignment	148
$Z \leftarrow \square TS$	Time Stamp	152
$Z \leftarrow \square AV$	Atomic Vector	153
$Z \leftarrow \square LC$	Line Counter	153
$Z \leftarrow \square DL B$	Delay	154
$Z \leftarrow \square NC B$	Name Class	155
$Z \leftarrow \square EX B$	Expunge	156
$Z \leftarrow \square NL B$	Name List	156
$Z \leftarrow \square STOP B$	Query Stop	157
$Z \leftarrow \square TRACE B$	Query Trace	157
$Z \leftarrow A \square NL B$	Name List	158
$Z \leftarrow A \square STOP B$	Set Stop	158
$Z \leftarrow A \square TRACE B$	Set Trace	159
$Z \leftarrow \square CT \leftarrow B$	Comparison Tolerance	162
$Z \leftarrow \square CT$	Comparison Tolerance	162
$Z \leftarrow \square RL \leftarrow B$	Random Link	163
$Z \leftarrow \square RL$	Random Link	163
$Z \leftarrow \square PP \leftarrow B$	Print Precision	164
$Z \leftarrow \square PP$	Print Precision	164
$Z \leftarrow \square IO \leftarrow B$	Index Origin	165
$Z \leftarrow \square IO$	Index Origin	165
$Z \leftarrow \square LX \leftarrow B$	Latent Expression	166
$Z \leftarrow \square LX$	Latent Expression	166

Table 4 — The Form Table (continued)

Form	Operation Name	Page
$Z \leftarrow \square LX[I] \leftarrow B$	Latent Expression	166
$Z \leftarrow DFN$	Call-Defined-Function	174
$Z \leftarrow DFN B$	Call-Defined-Function	174
$Z \leftarrow A DFN B$	Call-Defined-Function	174
$Z \leftarrow \square FX B$	Function Fix	176
$Z \leftarrow \square CR B$	Character Representation	177
$Z \leftarrow SHV$	Shared Variable Reference	189
$Z \leftarrow SHV \leftarrow B$	Shared Variable Assignment	189
$Z \leftarrow SHV[I] \leftarrow B$	Shared Variable Indexed Assignment	190
$Z \leftarrow \square SVC B$	Shared Variable Access Control Inquiry	191
$Z \leftarrow \square SVQ B$	Shared Variable Query	192
$Z \leftarrow \square SVO B$	Shared Variable Degree of Coupling	193
$Z \leftarrow A \square SVO B$	Shared Variable Offer	194
$Z \leftarrow \square SVR B$	Shared Variable Retraction	195
$Z \leftarrow A \square SVC B$	Shared Variable Access Control Set	196
$Z \leftarrow \Phi B$	Monadic Format	203
$Z \leftarrow A \Phi B$	Dyadic Format	206
$Z \leftarrow \square$	Quad Input	213
$Z \leftarrow \square$	Quote Quad Input	214
$Z \leftarrow \square \leftarrow B$	Quad Output	214
$Z \leftarrow \square \leftarrow B$	Quote Quad Output	215

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

7 SCALAR FUNCTIONS

Note: The primitive functions described in this chapter are called **scalar-functions**. All **scalar-functions** have uniform behaviour with respect to the structure of their argument arrays. The shape of the result of a **scalar-function** is determined solely by the shapes of its arguments.

This section defines **scalar-functions** individually for scalar arguments. Their common behaviour is described in this informal description by the expository device of an implicit operator, called the **scalar-extension-operator**.

If the argument to a monadic scalar function is not a scalar, a monadic scalar function extension operator can be thought of as being invoked to produce a derived function which, in turn, applies the monadic scalar function to every element of the argument array, producing a result array of the same shape as the argument. The order in which the elements of the argument array are presented to the monadic scalar function is not specified by this standard. Monadic scalar functions never signal **rank-error** or **length-error**.

If either of the arguments of a dyadic scalar function is not a scalar, a dyadic scalar extension operator can be thought of as being invoked to produce a derived function, which provides pairs of scalars to the scalar function as follows:

The dyadic scalar extension operator first tests whether the two argument arrays have the same shape. Arguments to a dyadic scalar function must have the same shape. If they do not, and the argument of lesser rank is a scalar or one-element vector, the argument of lesser rank is reshaped to the shape of the argument of greater rank.

If the arguments cannot be made to have the same shape, the dyadic scalar extension operator signals a **rank-error** if the arguments are of different ranks and a **length-error** otherwise.

When the dyadic scalar extension operator succeeds, it produces a derived function which generates a scalar for each position in its result array by applying the subject **scalar-function** to pairs of scalars selected from corresponding positions in the argument arrays. The order in which the elements of the result array are produced is not specified by this standard.

Because the derived function produced by either scalar extension operator never calls its **scalar-function** argument for empty arrays, **domain-error** can never be signalled for empty array arguments or for arrays reshaped by scalar extension to empty.

The type of all empty results produced by the functions derived from monadic and dyadic scalar extension is a property of the function argument to scalar extension. Since all scalar functions specified in this standard produce **numeric** results, the type of all empty results produced by scalar extension is specified as **numeric**.

For example, $''\wedge 5$ and $-''$ return empty **numeric** results rather than signalling an error.

7.1 MONADIC SCALAR FUNCTIONS

Note: The definitions in this section cover only scalar arguments. The phrase-evaluator `evaluate-monadic-function` handles non-scalar cases. All scalar-functions yield scalar results when applied to scalar arguments.

Note that, in this standard, `roll` is not a scalar-function.

7.1.1 Conjugate

$Z \leftarrow + B$

Informal Description: Z is B .

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
Return B .

Example:

$$\begin{array}{ccccccc} & & + & 3 & ^{-4} & 0 & 0.5 \\ 3 & ^{-4} & 0 & 0.5 & & & \end{array}$$

Note: This operation is named "conjugate" in accordance with its intended use in complex arithmetic.

7.1.2 Negative

$Z \leftarrow - B$

Informal Description: Z is the **negation** of B .

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
Return **zero** minus B .

Example:

$$\begin{array}{ccccccc} & & - & 7 & 0 & ^{-7} \\ ^{-7} & 0 & 7 & & & & \end{array}$$

7.1.3 Signum

$$Z \leftarrow \times B$$

Informal Description: Z is -1 , 0 , or 1 , according to whether B is negative, zero, or positive, respectively.

Evaluation Sequence:

- If B is not a **number**, signal **domain-error**.
- If B is **zero**, return **zero**.
- If B is **greater-than zero**, return **one**.
- If B is **less-than zero**, return **negative-one**.

Example:

$$\begin{array}{r} \times 1 \text{ } ^{-} .5 \text{ } .33 \text{ } 0 \text{ } ^{-} 1 \text{ } E^{-} 20 \\ 1 \text{ } ^{-} 1 \text{ } 1 \text{ } 0 \text{ } ^{-} 1 \end{array}$$

7.1.4 Reciprocal

$$Z \leftarrow \div B$$

Informal Description: Z is $1 \div B$.

Evaluation Sequence:

- If B is not a **number**, signal **domain-error**.
- If B is **zero**, signal **domain-error**.
- Return **one divided-by** B .

Examples:

$$\begin{array}{r} \div \text{ } ^{-} .25 \text{ } .5 \text{ } 1 \text{ } 2 \text{ } ^{-} 4 \\ ^{-} 4 \text{ } 2 \text{ } 1 \text{ } 0.5 \text{ } ^{-} 0.25 \end{array}$$

$\div 0$
domain-error

7.1.5 Floor

$$Z \leftarrow \lfloor B$$

Informal Description: Z is the greatest integer tolerantly less than or equal to B . Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is nil, signal **implicit-error**.
 If B is not a number, signal **domain-error**.
 Return the **tolerant-floor** of B within **comparison-tolerance**.

Example:

In the following, **comparison-tolerance** is $1E^{-10}$.

```

      L 3.1416 3.1416 .9999999999 5E20 0.5E-10
-4 3 1 5E20 0
    
```

7.1.6 Ceiling

$$Z \leftarrow \lceil B$$

Informal Description: Z is the least integer tolerantly greater than or equal to B . Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is nil, signal **implicit-error**.
 If B is not a number, signal **domain-error**.
 Return $-\lfloor -B$.

Example:

In the following, **comparison-tolerance** is $1E^{-10}$.

```

      3.1416 3.1416 5.00000000001
-3 4 5
    
```

7.1.7 Exponential

$$Z \leftarrow * B$$

Informal Description: Z is e raised to the power B , where e is the base of the natural logarithms.

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
Return the **exponential** of B .

Examples:

```

      * 1E50 -2 -1 0 1 2
0 0.135335 0.367879 1 2.71828 7.38906
      * .693147
2

```

7.1.8 Natural Logarithm

$$Z \leftarrow \otimes B$$

Informal Description: Z is the natural logarithm of B .

Evaluation Sequence:

If B is not a **positive-number**, signal **domain-error**.
Return the **natural-logarithm** of B .

Examples:

```

      \otimes 2.718281828459045 2 1E-50 1E50
1 0.693147 -115.129 115.129
      \otimes 1
1

```

7.1.9 Magnitude

$$Z \leftarrow | B$$

Informal Description: Z is the absolute value of B .

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
Return the **absolute-value** of B .

Example:

```

      | 1 -0.5 0.33 -0.25 0 1E-20
1 0.5 0.33 0.25 0 1E-20

```

7.1.10 Factorial

$$Z \leftarrow ! B$$

Informal Description: Z is the **gamma-function** of $B+1$. If B is a **nonnegative-integer**, this is factorial B .

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
If B is a **negative-integer**, signal **domain-error**.
Set $B1$ to B plus one.
Return **gamma-function** of $B1$.

Examples:

```

      ! 0 1 2 3 4 5 6 7 8 9
1 1 2 6 24 120 720 5040 40320 362880

      !-.5
1.77245

      5 1 p ! - 1.502 1.503 1.504 1.505 1.506
-3.54471
-3.54466
-3.54464
-3.54466
-3.5447

```

Note: The **gamma-function** is defined in, for example, the *National Bureau of Standards Handbook of Mathematical Functions*, U.S. Government Printing Office, Washington D.C., 1964.

See also Hart, J. F., *Computer Approximations*, Robert C. Krieger Publishing Company, Huntington, NY, 1978.

7.1.11 Pi times

$$Z \leftarrow \circ B$$

Informal Description: Z is π times B .

Evaluation Sequence:

If B is not a **number**, signal **domain-error**.
Return **pi-times** B .

Example:

```

      o 1 10 100
3.14159 31.4159 314.159

```

7.1.12 Not

$$Z \leftarrow \sim B$$

Informal Description: Z is the Boolean complement of B .

Evaluation Sequence:

If B is not **near-Boolean**, signal **domain-error**.
If the **integer-nearest-to** B is **one**, return **zero**.
Otherwise, return **one**.

Example:

For the following, the **implementation-parameter integer-tolerance** is $1E^{-10}$.

```

      ~ 0 1 1E-11 .999999999999
1 0 1 0

```

7.2 DYADIC SCALAR FUNCTIONS

Note: The definitions in this section cover only scalar arguments. The phrase-evaluator *evaluate-dyadic-function* handles non-scalar cases. All scalar-functions yield scalar results when applied to scalar arguments.

The outer product operator, which has not yet been formally introduced at this point in the document, is used in the examples in this section as a convenient way of generating tables. The use of outer product in this section is limited to vector arguments. The same results could be obtained from each example, although not so compactly, by supplying the elements of the left argument one at a time, starting from the leftmost, as left arguments to the scalar function.

For example,

$$\begin{array}{ccc} & 0 & 1 & 2 \\ 0 & 1 & 0 & . = 0 & 1 & 2 \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \end{array}$$

is equivalent to

$$\begin{array}{ccc} & 0 & = & 0 & 1 & 2 \\ 1 & 0 & 0 & \\ & 1 & = & 0 & 1 & 2 \\ 0 & 1 & 0 & \end{array}$$

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

7.2.1 Plus

$$Z \leftarrow A + B$$

Informal Description: Z is A plus B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
Return A plus B .

Example:

$$\begin{array}{cccc} & -2 & -1 & 0 & 1 \\ -4 & -3 & -2 & -1 & 0 \\ -3 & -2 & -1 & 0 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ -1 & 0 & 1 & 2 & 3 \end{array} \quad \circ . + \quad \begin{array}{cccc} & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \end{array}$$

7.2.2 Minus

$$Z \leftarrow A - B$$

Informal Description: Z is A minus B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
Return A minus B .

Example:

$$\begin{array}{cccc} & -2 & -1 & 0 & 1 \\ 0 & -1 & -2 & -3 & -4 \\ 1 & 0 & -1 & -2 & -3 \\ 2 & 1 & 0 & -1 & -2 \\ 3 & 2 & 1 & 0 & -1 \end{array} \quad \circ . - \quad \begin{array}{cccc} & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \\ & -2 & -1 & 0 & 1 \end{array}$$

7.2.3 Times

$$Z \leftarrow A \times B$$

Informal Description: Z is A times B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
Return A times B .

Example:

		⁻²	⁻¹	0	1	◦.×	⁻²	⁻¹	0	1
4	2	0	⁻²							
2	1	0	⁻¹							
0	0	0	0							
⁻²	⁻¹	0	1							

7.2.4 Divide

$$Z \leftarrow A \div B$$

Informal Description: Z is A divided by B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
If B is **zero** and A is not **zero**, signal **domain-error**.
If B is **zero** and A is **zero**, return **one**.
Otherwise, return A divided-by B .

Example:

		0	1	2	3	4	◦.÷	1	2	3	4
0	0	0				0					
1	0.5	0.333333	0.25								
2	1	0.666666	0.5								
3	1.5	1	0.75								
4	2	1.33333	1								

7.2.5 Maximum

$$Z \leftarrow A \uparrow B$$

Informal Description: Z is the larger of A and B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.

If A is **greater-than** B , return A .

Otherwise, return B .

Example:

$$\begin{array}{cccc} & -2 & -1 & 0 & 1 & \circ \cdot \uparrow & -2 & -1 & 0 & 1 \\ -2 & -1 & 0 & 1 & & & -2 & -1 & 0 & 1 \\ -1 & -1 & 0 & 1 & & & & & & \\ 0 & 0 & 0 & 1 & & & & & & \\ 1 & 1 & 1 & 1 & & & & & & \end{array}$$

7.2.6 Minimum

$$Z \leftarrow A \downarrow B$$

Informal Description: Z is the smaller of A and B .

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.

If A is **greater-than** B , return B .

Otherwise, return A .

Example:

$$\begin{array}{cccc} & -2 & -1 & 0 & 1 & \circ \cdot \downarrow & -2 & -1 & 0 & 1 \\ -2 & -2 & -2 & -2 & & & -2 & -1 & 0 & 1 \\ -2 & -1 & -1 & -1 & & & & & & \\ -2 & -1 & 0 & 0 & & & & & & \\ -2 & -1 & 0 & 1 & & & & & & \end{array}$$

7.2.7 Power

$$Z \leftarrow A * B$$

Informal Description: Z is A raised to the B th power.

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
 If A is a **negative-number** and B is not an **integer**, signal **domain-error**.
 If A is **zero** and B is a **negative-number**, signal **domain-error**.
 If A is **zero** and B is **zero**, return **one**.
 Return A **to-the-power** B .

Examples:

In the following, **print-precision** is 12.

$$2 * 32$$

4294967296

$$4 * 0.5$$

2

$$-8 * \div 3$$

domain-error

Note: The distinction made in some APL implementations between rational and irrational powers (to permit expressions such as $-8 * \div 3$ to have real results) has been eliminated in this standard to allow complex arithmetic to be a **consistent-extension**.

7.2.8 Logarithm

$$Z \leftarrow A \oslash B$$

Informal Description: Z is the logarithm of B to the base A .

Evaluation Sequence:

If A is not a **positive-number**, signal **domain-error**.
 If B is not a **positive-number**, signal **domain-error**.
 If A is **one** and B is **one**, return **one**.
 If A is **one**, signal **domain-error**.
 Set $A1$ to the **natural-logarithm** of A .
 Set $B1$ to the **natural-logarithm** of B .
 Return $B1$ **divided-by** $A1$.

Example:

$$10 \ 2 \ 10 \ 0.1 \ \oslash \ 2 \ 65536 \ 1E15 \ 1E15$$

0.30103 16 15 $^{-15}$

7.2.9 Residue

$$Z \leftarrow A \mid B$$

Informal Description: Z is B modulo A . Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is nil, signal **implicit-error**.
 If either of A or B is not a **number**, signal **domain-error**.
 If A is **zero**, return B .
 If **comparison-tolerance** is not **zero**, and B **divided-by** A is **integral-within** **comparison-tolerance**, return **zero**.
 Otherwise set Z to B **modulo** A .
 If Z is A , return **zero**.
 Otherwise, return Z .

Examples:

In the following, **print-precision** is 16 and **comparison-tolerance** is $1E^{-10}$.

```

      7 -7 0. | 31 28 -30
-3 0 5
-4 0 -2

```

```

      0.2. | 1.4 1.5 1.6
0 0.1 0

```

```

      1 | 1E30 1E-30 -1E-30 .999999999999
0 0 0 0

```

In the following, **comparison-tolerance** is **zero**.

```

      1 | 1E30 1E-30 -1E-30 .999999999999
0 1E-30 0 0.999999999999

```

Additional Requirements:

The range of residue is the union of **zero** and the **open-interval-between** **zero** and A , except when A is **zero**, in which case the range is the single **number** B .

Note: The implementation-algorithm P modulo Q provides an exact modulo operation. It evaluates $R \leftarrow P - (\times P) \times \lfloor Q \times \lfloor P \div Q \rfloor$ exactly, and returns R if $(\times R) = \times Q$, or $R + Q$ otherwise.

The definition of "mod" in the proposed IEEE standard for Binary Floating-Point Arithmetic (P754) provides an example of this exact evaluation.

Implementations should avoid signalling **limit-error** in residue. If the operation B **divided-by** A causes **exponent-overflow**, return **zero**. If it causes **exponent-underflow**, and if A and B have the same signs, return B . If they have different signs, return **zero**.

7.2.10 Binomial

$Z \leftarrow A \text{ ! } B$

Informal Description: Z is $(\text{gamma}(1+B)) \div ((\text{gamma}(1+A)) \times \text{gamma}(1+B-A))$

If A and B are **nonnegative-integers**, Z is the number of combinations of B things taken A at a time.

Evaluation Sequence:

If either of A or B is not a **number**, signal **domain-error**.
Determine if each of A , B , and $B-A$ is a **negative-integer**.
Select the appropriate case from the following table, where a **one** indicates that the corresponding value is a **negative-integer** and a **zero** indicates that it is not.

Case			Rule
A	B	$B-A$	
0	0	0	Return $(!B) \div (!A) \times !B-A$.
0	0	1	Return zero .
0	1	0	Signal domain-error .
0	1	1	Return $(-1 \times A) \times A ! A-B+1$.
1	0	0	Return zero .
1	0	1	(Case cannot arise.)
1	1	0	Return $(-1 \times B-A) \times (B+1) ! (A+1)$.
1	1	1	Return zero .

Example:

-4	-3	-2	-1	0	1	2	3	4	...	!	-4	-3	-2	-1	0	1	2	3	4
1	-3	3	-1	0	0	0	0	0	0										
0	1	-2	1	0	0	0	0	0	0										
0	0	1	-1	0	0	0	0	0	0										
0	0	0	1	0	0	0	0	0	0										
1	1	1	1	1	1	1	1	1	1										
-4	-3	-2	-1	0	1	2	3	4											
10	6	3	1	0	0	1	3	6											
-20	-10	-4	-1	0	0	0	1	4											
35	15	5	1	0	0	0	0	1											

Note: The APL expressions in the rule column indicate the result required, not the algorithm to be used. For example, $64!65$ should be 65 even if $!65$ signals **limit-error**.

7.2.11 Circular Functions

$$Z \leftarrow A \circ B$$

Informal Description: Z is the result of applying a function designated by A to B .

Evaluation Sequence:

If A is not a **near-integer**, signal **domain-error**.

If B is not a **number**, signal **domain-error**.

Set $A1$ to the **integer-nearest-to** A .

If $A1$ is not in the **closed-interval-between** -7 and 7 , signal **domain-error**.

If $A1$ is -7 ,

If B is not in the **open-interval-between negative-one and one**, signal **domain-error**.
Return the **inverse-hyperbolic-tangent** of B .

If $A1$ is -6 ,

If B is **less-than one**, signal **domain-error**.

Return Z , the principal value of the **inverse-hyperbolic-cosine** of B , where Z is a **nonnegative-number**.

If $A1$ is -5 , return the **inverse-hyperbolic-sine** of B .

If $A1$ is -4 ,

If B is in the **open-interval-between negative-one and one**, signal **domain-error**.
Return $B \times (1 - B^2)^{0.5}$.

If $A1$ is -3 , return Z , the principal value in radians of the **inverse-tangent** of B , where Z is in the **open-interval-between** $-\pi/2$ and $\pi/2$.

If $A1$ is -2 ,

If B is not in the **closed-interval-between negative-one and one**, signal **domain-error**.

Return Z , the principal value in radians of the **inverse-cosine** of B , where Z is either **zero** or a **number** in the **open-interval-between zero and π** .

If $A1$ is -1 ,

If B is not in the **closed-interval-between negative-one and one**, signal **domain-error**.

Return Z , the principal value in radians of the **inverse-sine** of B , where Z is either $\pi/2$, or a **number** in the **open-interval-between** $-\pi/2$ and $\pi/2$.

If $A1$ is 0,

If B is not in the **closed-interval-between negative-one and one**, signal **domain-error**.
Return $(1-B*2)*0.5$.

If $A1$ is 1, return the **sine** of B radians.

If $A1$ is 2, return the **cosine** of B radians.

If $A1$ is 3,

If B is an odd multiple of $\pi/2$, signal **domain-error**.
Return the **tangent** of B radians.

If $A1$ is 4, return $(1+B*2)*0.5$.

If $A1$ is 5, return the **hyperbolic-sine** of B .

If $A1$ is 6, return the **hyperbolic-cosine** of B .

If $A1$ is 7, return the **hyperbolic-tangent** of B .

Examples:

	2	o	⁻ 1	o	.6
.8	2	o	0		
1	3	oo	÷4		
1	6	o	0		
1					

Note: The APL expressions used for $0 \circ X$, $^{-}4 \circ X$, and $4 \circ X$ above indicate the result desired, not the algorithm to be used.

The definition of $^{-}4 \circ B$ has been changed to allow complex arithmetic to be a **consistent-extension**. The previous definition was $(^{-}1+B*2)*0.5$.

7.2.12 And

$$Z \leftarrow A \wedge B$$

Informal Description: Z is the Boolean product of A and B .

Evaluation Sequence:

If either A or B is not **near-Boolean**, signal **domain-error**.

Set $A1$ to the **integer-nearest-to** A .

Set $B1$ to the **integer-nearest-to** B .

If either $A1$ or $B1$ is **zero**, return **zero**.

Otherwise, return **one**.

Example:

```

      0 1 0 . ^ 0 1
0 0
0 1

```

7.2.13 Or

$$Z \leftarrow A \vee B$$

Informal Description: Z is the Boolean sum of A and B .

Evaluation Sequence:

If either A or B is not **near-Boolean**, signal **domain-error**.

Set $A1$ to the **integer-nearest-to** A .

Set $B1$ to the **integer-nearest-to** B .

If either $A1$ or $B1$ is **one**, return **one**.

Otherwise, return **zero**.

Example:

```

      0 1 0 . v 0 1
0 1
1 1

```

7.2.14 Nand

$$Z \leftarrow A \nabla B$$

Informal Description: Z is the Boolean complement of the Boolean product of A and B .

Evaluation Sequence:

If either A or B is not **near-Boolean**, signal **domain-error**.
Otherwise, return $\sim A \wedge B$.

Example:

	0	1	∘	⋅	⋈	0	1
1	1						
1	0						

7.2.15 Nor

$$Z \leftarrow A \nabla B$$

Informal Description: Z is the Boolean complement of the Boolean sum of A and B .

Evaluation Sequence:

If either A or B is not **near-Boolean**, signal **domain-error**.
Otherwise, return $\sim A \vee B$.

Example:

	0	1	∘	⋅	⋈	0	1
1	0						
0	0						

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

7.2.16 Equal

$$Z \leftarrow A = B$$

Informal Description: Z is **one** if A and B are considered equal and **zero** otherwise. A and B are equal if they are the same **character**, or if they are both **numeric** and A is tolerantly equal to B within **comparison-tolerance**. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.

If the **type** of A is not the same as the **type** of B , return **zero**.

If both A and B are **characters**,

If A is the same **character** as B , return **one**.

Otherwise, return **zero**.

If both A and B are **numbers**,

If A is **tolerantly-equal** to B within **comparison-tolerance**, return **one**.

Otherwise, return **zero**.

Example:

```

      1 2 3 0.= 1 2 3
1 0 0
0 1 0
0 0 1

```

In the following, **comparison-tolerance** is $1E^{-13}$.

```

      4 = 4 + 5E-13 2E-13 -2E-13 -5E-13
0 1 1 0

      0 = -1E-20 1E-20 0
0 0 1

      3 = 'A3'
0 0

```

Note: Comparisons of numbers whose signs differ are not affected by comparison-tolerance .

For any value of comparison-tolerance and any two values A and B exactly one of the expressions $A < B$, $A = B$, and $A > B$ is one.

Equal should not signal a limit-error. For example, the result of positive-number-limit = negative-number-limit is zero. The following is a sample technique for handling exponent-overflow and exponent-underflow when scaling comparison-tolerance .

Set C to the larger of the absolute values of A and B .

Set D to comparison-tolerance times C .

If exponent-underflow occurs,

Set $A1$ to A divided-by C .

Set $B1$ to B divided-by C .

Set $C1$ to the absolute value of $A1$ minus $B1$.

If $C1$ is greater-than comparison-tolerance , return zero.

Otherwise, return one.

Set E to the absolute value of A minus B .

If exponent-overflow occurs, return zero.

If exponent-underflow occurs,

Set $A1$ to A divided-by C .

Set $B1$ to B divided-by C .

Set $C1$ to the absolute value of $A1$ minus $B1$.

If $C1$ is greater-than comparison-tolerance , return zero.

Otherwise, return one.

If E is not greater-than D , return one.

Otherwise, return zero.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

7.2.17 Less than

$$Z \leftarrow A < B$$

Informal Description: Z is **one** if A is tolerantly less-than B , and **zero** otherwise.
Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.
If either of A or B is not a **number**, signal **domain-error**.
If $A=B$, evaluated with the current value of **comparison-tolerance**, is **one**, return **zero**.
If A is **less-than** B , return **one**.
Otherwise, return **zero**.

Examples:

```

      1 2 3 °.< 1 2 3
0 1 1
0 0 1
0 0 0

      0 1 °.< 0 1
0 1
0 0

```

7.2.18 Less than or equal to

$$Z \leftarrow A \leq B$$

Informal Description: Z is **one** if A is less than or tolerantly equal to B , and **zero** otherwise. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.
If either of A or B is not a **number**, signal **domain-error**.
If $A=B$, evaluated with the current value of **comparison-tolerance**, is **one**, return **one**.
If A is **less-than** B , return **one**.
Otherwise, return **zero**.

Examples:

```

      1 2 3 °.≤ 1 2 3
1 1 1
0 1 1
0 0 1

      0 1 °.≤ 0 1
1 1
0 1

```

7.2.19 Not equal

$$Z \leftarrow A \neq B$$

Informal Description: Z is **one** if A does not equal B , and **zero** otherwise. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.

Return $\sim A=B$, evaluated with the current value of **comparison-tolerance**.

Examples:

```

      'A' ≠ 41
1
1 2 3 0. ≠ 1 2 3
0 1 1
1 0 1
1 1 0

      0 1 0. ≠ 0 1
0 1
1 0
    
```

7.2.20 Greater than or equal to

$$Z \leftarrow A \geq B$$

Informal Description: Z is **one** if A is greater than or tolerantly equal to B , and **zero** otherwise. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.

If either of A or B is not a **number**, signal **domain-error**.

If $A=B$, evaluated with the current value of **comparison-tolerance**, is **one**, return **one**.

If A is **greater-than** B , return **one**.

Otherwise, return **zero**.

Examples:

```

      1 2 3 0. ≥ 1 2 3
1 0 0
1 1 0
1 1 1

      0 1 0. ≥ 0 1
1 0
1 1
    
```

7.2.21 Greater than

$$Z \leftarrow A > B$$

Informal Description: Z is **one** if A is tolerantly greater than B , and **zero** otherwise. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.

If either of A or B is not a **number**, signal **domain-error**.

If $A=B$, evaluated with the current value of **comparison-tolerance**, is **one**, return **zero**.

If A is **greater-than** B , return **one**.

Otherwise, return **zero**.

Examples:

```

      1 2 3 °.> 1 2 3
0 0 0
1 0 0
1 1 0

```

```

      0 1 °.> 0 1
0 0
1 0

```

STANDARDISO.COM : Click to view the full PDF of ISO 8485:1989

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

8 STRUCTURAL PRIMITIVE FUNCTIONS

8.1 INTRODUCTION

Note: The functions in this chapter are used in the evaluation sequences of many non-scalar operations. They are defined here to avoid forward references.

8.2 MONADIC STRUCTURAL PRIMITIVE FUNCTIONS

8.2.1 Ravel

$$Z \leftarrow , B$$

Informal Description: Z is a vector containing the elements of B in row-major order.

Evaluation Sequence:

Return Z , a **vector** such that the **ravel-list** of Z is the same as the **ravel-list** of B , the **type** of Z is the same as the **type** of B , and the **shape-list** of Z is a **list** of length **one** containing the **count** of B as its only **item**.

Examples:

```

      ,N22
11 12 21 22
      ,N222
111 112 121 122 211 212 221 222
      ,N2221
1111 1121 1211 1221 2111 2121 2211 2221

```

Note: *Ravel* always produces a vector result. The expressions **ravel-list** of Z , **type** of Z , and **shape-list** of Z refer to attributes of an array object.

8.2.2 Shape

$$Z \leftarrow \rho B$$

Informal Description: Z is a **numeric** vector containing the shape of the array B .

Evaluation Sequence:

Return Z , an **array** such that the **type** of Z is **numeric**, the **ravel-list** of Z is the **shape-list** of B , and the **shape-list** of Z is a **list** whose only **item** contains the **number-of-items** in the **shape-list** of B .

Examples:

	ρN
	ρ, N
1	$\rho \rho N$
0	$\rho N 3$
3	$\rho \rho N 3$
1	$\rho N 3 4$
3 4	

Note: *Shape always produces a vector result. The expression **shape-list** of Z refers to an attribute of an array object.*

8.2.3 Index Generator

$$Z \leftarrow 1 \ B$$

Informal Description: Z is a **numeric** vector of B consecutive ascending **integers**, the first of which is **index-origin**. Uses **index-origin**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is not **one**, signal **length-error**.

If B is not a **near-integer**, signal **domain-error**.

Set $B1$ to the **integer-nearest-to** B .

If $B1$ is not a **nonnegative-integer**, signal **domain-error**.

If $B1$ is **zero**, return an **empty numeric vector**.

Generate a **numeric vector** $Z1$ of length $B1$ such that the **ravel-list** of $Z1$ consists of the **integers** in the **closed-interval-between** **one** and $B1$ in ascending order.

Generate Z , a **numeric array** with the **shape-list** of $Z1$ such that each **item** of the **ravel-list** of Z is (**index-origin** minus **one**) plus the corresponding element of $Z1$.

Return Z .

Examples:

In the following, **index-origin** is **zero**.

$$\begin{array}{cccc} & & 1 & 4 \\ 0 & 1 & 2 & 3 \end{array}$$

In the following, **index-origin** is **one**.

$$\begin{array}{cccc} & & 1 & 4 \\ 1 & 2 & 3 & 4 \end{array}$$

8.3 DYADIC STRUCTURAL PRIMITIVE FUNCTIONS

8.3.1 Reshape

$$Z \leftarrow A \rho B$$

Informal Description: Z is an array of shape $,A$ whose elements are taken sequentially from $,B$ repeated cyclically as required.

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.

If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.

Set $A1$ to the **integer-array-nearest-to** $,A$.

If any **item** of the **ravel-list** of $A1$ is not a **nonnegative-counting-number**, signal **domain-error**.

Let RA stand for the **product-of** the **ravel-list** of $A1$.

Let CB stand for the **count** of B .

If RA is not **zero** and CB is **zero**, signal **length-error**.

Return an **array** Z such that the **type** of Z is the same as the **type** of B , the **shape-list** of Z is the same as the **ravel-list** of $A1$, and the **ravel-list** of Z is a **list** with RA **items** such that for all I in the **index-set** of Z , **item** I of the **ravel-list** of Z is **item** $1+CB \div I - 1$ of the **ravel-list** of B .

Examples:

```

      2 4pN213
111 112 113 211
212 213 111 112
      p0p'ABCD'
0
      B ← 1E7 1E7 1E7 0 1E7 1E7 1E7 p 42
      pB
1E7 1E7 1E7 0 1E7 1E7 1E7

```

Note: For any X that is not empty, $1 \uparrow pX$ and $(1\ 0) \rho X$ produce the same result: a **scalar** whose value is that of the **first-scalar** in X .

8.3.2 Join

$$Z \leftarrow A , B$$

Informal Description: If A and B are scalars or vectors, Z is the vector of length $(\rho A) + \rho B$ whose first ρA elements are A and whose last ρB elements are B .

If either A or B has rank greater than one, Z is $A, [(\rho A) \uparrow \rho B] B$, as defined under **Join Along an Axis**.

Evaluation Sequence:

If A is a **scalar** and B is a **scalar**, return $(, A) , , B$.

If A is a **scalar** and B is a **vector**, return $(, A) , B$.

If A is a **vector** and B is a **scalar**, return $A , , B$.

If A is a **vector** and B is a **vector**,

If A is **empty** and B is **empty** and the **type** of A differs from the **type** of B , signal **domain-error**.

If B is **empty**, return A .

If A is **empty**, return B .

If the **type** of A differs from the **type** of B , signal **domain-error**.

Otherwise, return a **vector** Z , such that the **shape-list** of Z is $(\rho A) + \rho B$, the **ravel-list** of Z is a **list** whose first ρA **items** are the **ravel-list** of A and whose last ρB **items** are the **ravel-list** of B , and the **type** of Z is the same as the **type** of A .

Otherwise, return $A , [(\rho A) \uparrow \rho B] B$.

Example:

0 1 1 , 0

Note: This subsection intentionally contains a forward reference to **Join Along an Axis**. **Join** and **Join Along an Axis** are defined separately because the description of **Join Along an Axis** requires APL operations that depend for their definitions in turn upon **Join**.

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

9 OPERATORS

9.1 INTRODUCTION

Note: The forms in this chapter are referred to as **operators**.

Operators take scalar functions as arguments and produce functions, called derived functions, as results. For example, the operator **reduction** takes a single dyadic **scalar-function** as an argument and produces a monadic function as a result.

Because f is restricted to scalar functions, the derived functions produced by scan and reduction may be viewed either as being applied between subarrays of rank $0 \leq p \leq B$ or as being repeated on vector subarrays. As an expository device, the evaluation sequences in this document use the vector definition.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

9.2 MONADIC OPERATORS

9.2.1 Reduction

$Z \leftarrow f / B$
 $Z \leftarrow f / [K] B$
 $Z \leftarrow f \neq B$
 $Z \leftarrow f \neq [K] B$

Informal Description: Z is the value produced by placing the primitive scalar dyadic function f between subarrays of B and evaluating the resulting expression. The axis designated determines how the subarrays are chosen. Uses **index-origin**.

Evaluation Sequence:

If f is not a **primitive-dyadic-scalar-function**, signal **syntax-error**.

For form f/B

If B is a **scalar**, return B .
 Otherwise, return $f/[p p B] B$ evaluated with **index-origin** set to **one**.

For form $f \neq B$

If B is a **scalar**, return B .
 Otherwise, return $f/[1] B$ evaluated with **index-origin** set to **one**.

For forms $f/[K] B$ and $f \neq [K] B$

If K is not a **valid-axis** for B , signal **axis-error**.
 Otherwise, set $K1$ to the **integer-nearest-to** K .
 If B is a **vector**,

If the **length** of B is **zero**, take the action designated in **Table 5** for f .
 If the **length** of B is **one**, return a **scalar** Z such that the **type** of Z is the **type** of B and the **ravel-list** of Z is the **ravel-list** of B .
 If the **length** of B is **greater-than one**,

Set $B1$ to the **first-scalar** in B .
 Set $B2$ to the **remainder-of** B .
 Return $B1 f f/B2$.

If the **rank** of B is **greater-than one**, return an **array** Z such that the **shape-list** of Z is the **shape-list** of B with **item** $K1$ omitted, and the **ravel-list** of Z has the property that if $Z1$ is an **item** of Z and $B3$ is the corresponding **vector-item along-axis** $K1$ of B , then $Z1$ is $f/B3$.

Examples:

	$+ / 1 \ 2 \ 3$
6	$\times / 1 \ 2$
2	$= / 'A'$
A	$= / 'AA'$
1	$= / 'AAA'$

0

Additional Requirements:

If Z is **empty**, the **type** of Z is determined by the argument function f ; since f is a scalar function in this standard, the **type** of Z when **empty** is always **numeric**.

The **scalar-function** f may signal **domain-error**.

When applied to a rank N array, the derived function produces a result array of rank $0 \leq N-1$. When applied along an **empty axis** in an array, reduction produces an array whose shape is that of the argument array with the designated **axis** deleted. The elements of the array, if any, are determined by the argument function and **Table 5**. In some cases, an error is signalled.

For example,

```

0 0      +/2 0 p5.1
2      p+/2 0p5.1

```

Table 5 — Actions for the Reduction of an Empty Vector

Dyadic Function		Action
Plus	+	Return zero .
Minus	-	Return zero .
Times	x	Return one .
Divide	÷	Return one .
Residue		Return zero .
Minimum	L	Return positive-number-limit .
Maximum		Return negative-number-limit .
Power	*	Return one .
Logarithm	⊗	Signal domain-error .
Circular	○	Signal domain-error .
Binomial	⋮	Return one .
And	^	Return one .
Or	∨	Return zero .
Nand	⌘	Signal domain-error .
Nor	⌘	Signal domain-error .
Less	<	Return zero .
Not greater	≤	Return one .
Equal	=	Return one .
Not less	≥	Return one .
Greater	>	Return zero .
Not equal	≠	Return zero .

9.2.2 Scan

$$\begin{aligned} Z &\leftarrow f \backslash B \\ Z &\leftarrow f \backslash [K] B \\ Z &\leftarrow f \backslash B \\ Z &\leftarrow f \backslash [K] B \end{aligned}$$

Informal Description: Z is an array having the same shape as B and containing the results produced by f reduction over all prefixes of a designated axis of B . Uses **index-origin**.

Evaluation Sequence:

If f is not a **primitive-dyadic-scalar-function**, signal **syntax-error**.

For form $f \backslash B$

If B is a **scalar**, return B .

Otherwise, return $f \backslash [\rho p B] B$ evaluated with **index-origin** set to **one**.

For form $f \backslash B$

If B is a **scalar**, return B .

Otherwise, return $f \backslash [1] B$ evaluated with **index-origin** set to **one**.

For forms $f \backslash [K] B$ and $f \backslash [K] B$

If K is not a **valid-axis** for B , signal **axis-error**.

Otherwise, set $K1$ to the **integer-nearest-to** K .

If B is a **vector**,

If the **count** of B is **less-than two**, return B .

If the **type** of B differs from the **type** of $f/B[12]$, signal **domain-error**.

Otherwise return Z , a **vector** such that the **type** of Z is the **type** of $f/B[12]$, the **shape-list** of Z is the **shape-list** of B , and the **ravel-list** of Z is such that item I of the **ravel-list** of Z is $f/B[1I]$ for all I in the **index-set** of the **ravel-list** of B .

If the **rank** of B is **greater-than one**, each **vector-item along-axis** $K1$ of Z is $f \backslash B1$, where $B1$ is the corresponding **vector-item along-axis** $K1$ of B .

Examples:

```

+ \ 1 1 1
1 2 3
^ \ 1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0
- \ 'A'
A
= \ 'AB'
domain-error

```

Additional Requirements:

If the operator **reduction** signals an error when called by **scan**, **scan** returns the resultant error **token**.

The evaluation sequence above describes a quadratic algorithm for **scan**. If the function f is associative, **scan** may be implemented with a linear algorithm.

*Note: Various error checks have been performed on K by the phrase evaluators, and **index-origin** is **one**, before this evaluation sequence is called.*

9.3 DYADIC OPERATORS

9.3.1 Outer Product

$$Z \leftarrow A \circ . f B$$

Informal Description: Z is an array of shape $(\rho A), \rho B$. The elements of Z are the result of applying the **primitive-dyadic-scalar-function** f to every possible combination of scalar arguments where the left argument is an element of A and the right an element of B .

Z is such that if I is an **index-list** that selects a single element of Z , the first ρA items of I are the **index-list** that would select from A the element used as the left argument to f and the last ρB items of I are the **index-list** that would select from B the element used as the right argument to f .

Evaluation Sequence:

If f is not a **primitive-dyadic-scalar-function**, signal **syntax-error**.

Return Z , an **array** such that the **type** of Z is **numeric**, the **shape-list** of Z is $(\rho A), \rho B$, and the **ravel-list** of Z has the following property:

Let I stand for an **item** of the **index-set** of the **ravel-list** of A .
 Let J stand for an **item** of the **index-set** of the **ravel-list** of B .
 Let X stand for **item** I of the **ravel-list** of A .
 Let Y stand for **item** J of the **ravel-list** of B .
 Let N stand for the **count** of B .
 Let P stand for $J + (N \times (I - 1))$.

Then, **item** P of the **ravel-list** of Z is $X f Y$.

Example:

```

      10 20 30 ◦.+ 1 2 3
11 12 13
21 22 23
31 32 33
```

Additional Requirements:

If the **scalar-function** f signals an error, outer product returns the resulting error token.

Note: The type of the result of outer product is numeric only because all permitted argument functions return numeric results.

9.3.2 Inner Product

$$Z \leftarrow A \text{ f } . \text{ g } B$$

Informal Description: Z is an array of shape $(\rho A)[10\overline{1}+\rho\rho A], (\rho B)[1+10\overline{1}+\rho\rho B]$. The elements of Z are the results obtained from evaluating the expression f/XgY for all possible combinations of X and Y , where X is a **vector-item along-axis** $\rho\rho A$ of A and Y a **vector-item along-axis one** of B .

Evaluation Sequence:

If either **f** or **g** is not a **primitive-dyadic-scalar-function**, signal **syntax-error**.

If A is a **scalar** or **one-element-vector** and B is not, set $A1$ to $(1\rho\rho B)\rho A$.

If A and B are **scalars** or **one-element-vectors**, set $A1$ to $,A$.

Otherwise, set $A1$ to A .

If B is a **scalar** or **one-element-vector** and A is not, set $B1$ to $(\rho A)[\rho\rho A]\rho B$.

If A and B are **scalars** or **one-element-vectors**, set $B1$ to $,B$.

Otherwise, set $B1$ to B .

If the **last-item** in the **shape-list** of $A1$ is not the same as the **first-item** in the **shape-list** of $B1$, signal **length-error**.

If $A1$ and $B1$ are both **vectors**, return $f/A1 \text{ g } B1$.

Otherwise, return Z , an **array** such that the **type** of Z is **numeric**, the **shape-list** of Z is $(\rho A1)[10\overline{1}+\rho\rho A1], (\rho B1)[1+10\overline{1}+\rho\rho B1]$ and the **ravel-list** of Z has the following property:

Let I stand for an item of the **index-set** of the **ravel-along-axis** $(\rho\rho A1)$ of $A1$.

Let X stand for **vector-item** I of the **ravel-along-axis** $(\rho\rho A1)$ of $A1$.

Let J stand for an item of the **index-set** of the **ravel-along-axis one** of $B1$.

Let Y stand for **vector-item** J of the **ravel-along-axis one** of $B1$.

Let N stand for the **number-of-items** in the **ravel-along-axis one** of $B1$.

Let P stand for $(N \times (I - 1)) + J$.

Then, **item** P of the **ravel-list** of Z is $f / X \text{ g } Y$.

Examples:

```

      4 2 1+.x1 0 1
5
      N22+.x0 1
12 22
      N22+.x1 0
11 21
      N22+.x2 2p0 1 1 0
12 11
22 21

```

Additional Requirements:

If **scalar-function** **f** or **g** signals an error, inner product returns the resulting error token.

Note: The evaluation sequence rule for when $A1$ and $B1$ are **vectors** holds if $A1$ or $B1$ is the empty vector. The result returned is $f/10$.

The type of the result of inner product is numeric only because all permitted argument functions return numeric results.

10 MIXED FUNCTIONS

10.1 MONADIC MIXED FUNCTIONS

10.1.1 Roll

$$Z \leftarrow ? B$$

Informal Description: Z , for B a scalar integer, is **index-origin** plus R , where R is an integer selected pseudorandomly from the set of all nonnegative integers less than B . Each integer in the set has an equal chance of being selected. Uses **index-origin**. Uses and sets **random-link**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.
 If **random-link** is **nil**, signal **implicit-error**.
 Let J stand for item I of the **ravel-list** of B .
 If B is not **scalar**, return a **numeric array** Z such that the **shape-list** of Z is the **shape-list** of B and, for all I in the **index-set** of the **ravel-list** of Z , item I of the **ravel-list** of Z is $?J$.
 If B is not a **near-integer**, signal **domain-error**.
 Set $B1$ to the **integer-nearest-to** B .
 If $B1$ is not a **positive-integer**, signal **domain-error**.
 Set $Z0$ to a **nonnegative-integer less-than** $B1$ and set **random-link** to a new value resulting from the **implementation-algorithm pseudorandom-number-generator** whose only inputs are $B1$ and **random-link**.
 Return $Z0$ plus **index-origin**.

Examples:

```
?1 1 1
1 1 1

S ← □RL
?4p1E30
1.69584E29 2.5949E29 5.38382E29 8.38274E29
□RL ← S
?4p1E30
1.69584E29 2.5949E29 5.38382E29 8.38274E29

(?1E20 1E20) ÷ 1E20
0.218959 0.678865
```

Additional Requirements:

The operation of **roll** is **atomic**: If **roll** signals an error, **random-link** shall be unchanged.

The result of $?B$, where B is an **array**, shall be reproducible.

A **conforming-implementation** shall provide documentation describing the properties of its **pseudorandom-number-generator**.

*Note: One class of appropriate algorithms is Lehmer's linear congruential method, described in Knuth, D. E., **Seminumerical Algorithms**, page 9.*

Roll is often considered a scalar function. However, it does not have the property that the elements of its result array can be produced in parallel.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

10.1.2 Grade Up

$$Z \leftarrow \Delta B$$

Informal Description: Z is, for B a vector, a permutation of $\iota p B$ for which $B[Z]$ is a monotone increasing sequence. The indices of identical elements of B occur in Z in ascending order. Uses **index-origin**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.

If the **rank** of B is not **one**, signal **rank-error**.

If pB is **zero**, return $\iota 0$.

If any **item** of the **ravel-list** of B is not a **number**, signal **domain-error**.

If pB is **one**, return a **one-element-vector** Z such that the **type** of Z is **numeric** and the **ravel-list** of Z contains **index-origin**.

Otherwise, generate $Z1$, a permutation of $\iota p B$ such that for I and J , arbitrary elements of $\iota p B$ for which I is **less-than** J ,

$B[Z1[I]]$ is not **greater-than** $B[Z1[J]]$ and

$B[Z1[I]]$ **equals** $B[Z1[J]]$ implies that $Z1[I]$ is **less-than** $Z1[J]$.

Generate Z , a **numeric array** with the **shape-list** of $Z1$ such that each **item** of the **ravel-list** of Z is (**index-origin** minus **one**) plus the corresponding element of $Z1$.

Return Z .

Examples:

```

      ΔV ← 1.1 3.1 1.1 2.1 5.1
1 3 4 2 5
      ΔΔV
1 4 2 3 5
      V[ΔV]
1.1 1.1 2.1 3.1 5.1

```

In the following, **index-origin** is **zero**.

```

      ΔV
0 2 3 1 4

```

Additional Requirements:

The **system-parameter** **comparison-tolerance** is not an implicit argument of **grade up**.

10.1.3 Grade Down

$$Z \leftarrow \Psi B$$

Informal Description: Z is, for a vector B , a permutation of $1 \text{ p } B$ for which $B[Z]$ is a monotone decreasing sequence. The indices of identical elements of B occur in Z in ascending order. Uses **index-origin**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.
 If the **rank** of B is not **one**, signal **rank-error**.
 Return $\Delta - B$.

Examples:

```

      V ← 1.1  3.1  1.1  2.1  5.1
      ΨV
5 2 4 1 3
      V[ΨV]
5.1 3.1 2.1 1.1 1.1
    
```

In the following, **index-origin** is **zero**.

```

      ΨV
4 1 3 0 2
    
```

Additional Requirements:

The **system-parameter comparison-tolerance** is not an implicit argument of grade down.

Note: Grade up and grade down are **stable sort algorithms** because they preserve the relative order of identical elements of B .

One appropriate algorithm for grade up and down appears in Woodrum, L. J., *Internal Sorting with Minimal Comparing*, IBM System Journal, Vol. 8, No. 3, p.189, 1969.

10.1.4 Reverse

$$\begin{aligned} Z &\leftarrow \phi B \\ Z &\leftarrow \theta B \\ Z &\leftarrow \phi[K] B \\ Z &\leftarrow \theta[K] B \end{aligned}$$

Informal Description: Z is an array whose elements are those of B taken in reverse order along a specified axis. Uses **index-origin**.

Evaluation Sequence:

For form ϕB

If B is **scalar**, return B .

Otherwise, return $\phi[\rho\rho B] B$, evaluated with **index-origin** set to **one**.

For form θB

If B is **scalar**, return B .

Otherwise, return $\theta[1] B$, evaluated with **index-origin** set to **one**.

For forms $\phi[K] B$ and $\theta[K] B$

If K is not a **valid-axis** for B , signal **axis-error**.

Otherwise, set $K1$ to the **integer-nearest-to** K .

If B is a **vector**, return $B[(1+\rho B)-1\rho B]$, evaluated with **index-origin** set to **one**.

Otherwise, return an **array** Z , such that the **type** of Z is the **type** of B , the **shape-list** of Z is the **shape-list** of B , and the **ravel-list** of Z has the property that each **vector-item along-axis** $K1$ of Z is ϕ applied to the corresponding **vector-item along-axis** $K1$ of B .

Examples:

```

       $\phi N23$ 
13 12 11
23 22 21
       $\phi[2] N224$ 

      121 122 123 124
      111 112 113 114

      221 222 223 224
      211 212 213 214

```

Note: Various error checks have been performed on K by the phrase evaluators, and **index-origin** is **one**, before this evaluation sequence is called.

10.1.5 Monadic Transpose

$$Z \leftarrow \mathbb{Q} B$$

Informal Description: Z is B with the order of the axes reversed.

Evaluation Sequence:

Return $(\phi_1 \rho \rho B) \mathbb{Q} B$, evaluated with **index-origin** set to **one**.

Examples:

	$\mathbb{Q} 3$
3	
	$\rho \rho \mathbb{Q} 3$
0	
	$\mathbb{Q} N23$
11	21
12	22
13	23
	$\mathbb{Q} N234$
111	211
121	221
131	231
112	212
122	222
132	232
113	213
123	223
133	233
114	214
124	224
134	234

Note: This subsection contains a forward reference to **dyadic transpose**.

10.1.6 Matrix Inverse

$$Z \leftarrow \boxminus B$$

Informal Description: Z is the result of applying a generalisation of the matrix inverse function to B .
Matrix inverse is **matrix divide** with an appropriate identity matrix as a left argument.

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Return $((2\rho 1\rho\rho B)\rho 1, (1\rho\rho B)\rho 0) \boxminus B$.

Note: This subsection contains a forward reference to **matrix divide**.

The following article, Martin, G. A., *The Solutions of Linear Systems in APL: Towards An Extension of Matrix Divide* APL80 Noordwijkerhout June 24-26, 1980 — Published by Gijsbert van der Linden, North-Holland Publishing Company Amsterdam. New York. Oxford, describes the motivation for matrix inverse and an acceptable algorithm.

The following document also deals with this subject: Jenkins, M. A., *Domino — An APL Primitive Function for Matrix Inversion — Its Implementation and Applications*. APL Quote-Quad Vol III No. 4, February 1972, pp. 4-15.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

10.1.7 Execute

$$Z \leftarrow \Phi B$$

Informal Description: Z is the result of evaluating the **character** scalar or vector B as a line of APL.

Evaluation Sequence:

If the **rank** of B is **greater-than one**, signal **rank-error**.

If any **item** of the **ravel-list** of B is not a **character**, signal **domain-error**.

Generate a new **context** in which

mode is **execute**,

current-line is the **ravel-list** of B ,

current-function is 0 0p ' ',

current-line-number is **one**.

current-statement is the empty list of tokens, and

stack is the empty list of tokens.

Append the new **context** to the **state-indicator** of the **active-workspace** as a new first item.

Set Z to **evaluate-line**.

Remove the first **context** from the **state-indicator**.

Return Z .

Examples:

$$\Phi 'T \leftarrow 3'$$

$$T$$

3

$$\square \leftarrow \Phi 'T \leftarrow 3'$$

3

$$A \leftarrow \Phi ' '$$

value-error

Note: If an error is signalled during **execute**, the user should be able to determine from information provided by the system where the error occurred in the argument of **execute** as well as where the failing **execute** primitive occurred in the **immediate-execution** or **defined-function** line.

10.2 DYADIC MIXED FUNCTIONS

10.2.1 Join Along an Axis

$$Z \leftarrow A, [K] B$$

Informal Description: Z is formed by joining A and B along a designated axis. There are two suboperations, catenate and laminate. The catenate suboperation joins the arrays along an existing axis, the laminate suboperation along a new axis. The choice of axis and the choice of operation is determined by K : if K is a **near-integer**, the operation is catenate and the axis is K ; if K is not a **near-integer**, the operation is laminate, the new axis is $\lceil K$, and the axes greater than or equal to $\lceil K$ are renumbered. Uses **index-origin**.

Evaluation Sequence:

If K is not a **near-integer**,

If A is a **scalar**, set $A1$ to $(\rho B)\rho A$.
Otherwise, set $A1$ to A .

If B is a **scalar**, set $B1$ to $(\rho A)\rho B$.
Otherwise, set $B1$ to B .

If the **rank** of $A1$ differs from the **rank** of $B1$, signal **rank-error**.

If K is not in the **open-interval-between zero** and **(one plus the rank of $A1$)**, signal **axis-error**.

If the **shape-list** of $A1$ differs from the **shape-list** of $B1$, signal **length-error**.

Set T to $(1, \rho A1)[\Delta K, \rho \rho A1]$.

Set $A2$ to $T\rho A1$.

Set $B2$ to $T\rho B1$.

Return $A2, [\lceil K \rceil B2$, evaluated with **index-origin** set to **one** and with **comparison-tolerance** set to **integer-tolerance**.

10.2 DYADIC MIXED FUNCTIONS

10.2.1 Join Along an Axis

$$Z \leftarrow A, [K] B$$

Informal Description: Z is formed by joining A and B along a designated axis. There are two suboperations, catenate and laminate. The catenate suboperation joins the arrays along an existing axis, the laminate suboperation along a new axis. The choice of axis and the choice of operation is determined by K : if K is a **near-integer**, the operation is catenate and the axis is K ; if K is not a **near-integer**, the operation is laminate, the new axis is $\lceil K$, and the axes greater than or equal to $\lceil K$ are renumbered. Uses **index-origin**.

Evaluation Sequence:

If K is not a **near-integer**,

If A is a **scalar**, set $A1$ to $(\rho B)\rho A$.
Otherwise, set $A1$ to A .

If B is a **scalar**, set $B1$ to $(\rho A)\rho B$.
Otherwise, set $B1$ to B .

If the **rank** of $A1$ differs from the **rank** of $B1$, signal **rank-error**.

If K is not in the **open-interval-between zero** and **(one plus the rank of $A1$)**, signal **axis-error**.

If the **shape-list** of $A1$ differs from the **shape-list** of $B1$, signal **length-error**.

Set T to $(1, \rho A1)[\Delta K, 1 \rho \rho A1]$.

Set $A2$ to $T\rho A1$.

Set $B2$ to $T\rho B1$.

Return $A2, [\lceil K \rceil] B2$, evaluated with **index-origin** set to **one** and with **comparison-tolerance** set to **integer-tolerance**.

Examples:

$\square \leftarrow M \leftarrow 2 \ 3p \ ' \Delta \'$
 $\Delta \Delta \Delta$
 $\Delta \Delta \Delta$
 $\square \leftarrow H \leftarrow 3 \ 3p \ 'o \'$
 ooo
 ooo
 ooo
 $M, [1]H$
 $\Delta \Delta \Delta$
 $\Delta \Delta \Delta$
 ooo
 ooo
 ooo
 $\square \leftarrow L \leftarrow 2 \ 4p \ ' \square \'$
 $\square \square \square \square$
 $\square \square \square \square$
 M, L
 $\Delta \Delta \Delta \square \square \square \square$
 $\Delta \Delta \Delta \square \square \square \square$
 $M, '+'$
 $\Delta \Delta \Delta +$
 $\Delta \Delta \Delta +$
 $M, '34'$
 $\Delta \Delta \Delta 3$
 $\Delta \Delta \Delta 4$
 $M, [1] '345'$
 $\Delta \Delta \Delta$
 $\Delta \Delta \Delta$
 345
 $1 \ 2 \ 3, [.5] \ 4 \ 5 \ 6$
 $1 \ 2 \ 3$
 $4 \ 5 \ 6$
 $1 \ 2 \ 3, [1.5] \ 4 \ 5 \ 6$
 $1 \ 4$
 $2 \ 5$
 $3 \ 6$
 $1 \ 2 \ 3, [1.5] \ 4$
 $1 \ 4$
 $2 \ 4$
 $3 \ 4$
 $(2 \ 0p5), 'A'$
 A
 A
 $p3, [.5] \ ' \'$
 $2 \ 0$

Note: Various error checks have been performed on K by the phrase evaluators, and index-origin is one, before this evaluation sequence is called.

10.2.2 Index of

$$Z \leftarrow A \uparrow B$$

Informal Description: Z is a numeric array of shape ρB . Each element of Z is the least index in A of a value **tolerantly-equal** within **comparison-tolerance** to the corresponding item of B . Uses **index-origin** and **comparison-tolerance**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.

If **comparison-tolerance** is **nil**, signal **implicit-error**.

If A is not a **vector**, signal **rank-error**.

Set Z to $+\wedge\backslash B \circ . \neq A$, evaluated with the current value of **comparison-tolerance**.

Return Z plus **index-origin**.

Examples:

In the following, **index-origin** is **zero**.

```

      ⍳←A←2 2p1.1 3.1 5.1 4.1
1.1 3.1
5.1 4.1
      3.1 4.1 5.1⍳A
3 0
2 1
      'ABC'⍳3
3

```

In the following, **index-origin** is **one**.

```

      '123ABC'⍳'3BD'
3 5 7
      '123ABC'⍳3
7

```

10.2.3 Member of

$$Z \leftarrow A \in B$$

Informal Description: Z is a Boolean array with the shape of A . An element of Z is **one** if the corresponding element of A is **tolerantly-equal** to some element of B ; otherwise, it is **zero**. Uses **comparison-tolerance**.

Evaluation Sequence:

If **comparison-tolerance** is **nil**, signal **implicit-error**.

Return $\vee / A \circ . = , B$, evaluated with the current value of **comparison-tolerance**.

Examples:

```

      B ← 2 2 p 1.1 3.1 5.1 4.1
1.1 3.1
5.1 4.1
      3.1 5.1 7.1 ∈ B
1 1 0
      19 ∈ 'CLUB'
0
      'BE' ∈ 'BOP'
1 0
      'NADA' ∈ 10
0 0 0 0
      (Γ/10) ∈ L/10
0

```

Note: If B is empty, the result of $A \in B$ is $(pA) p 0$.

10.2.4 Deal

$Z \leftarrow A ? B$

Informal Description: Z is **index-origin** plus R , where R is a vector of shape A obtained by making A pseudorandom selections without replacement from the set of nonnegative integers less than B . Uses **index-origin**. Uses and sets **random-link**.

Evaluation Sequence:

If **index-origin** is nil, signal **implicit-error**.

If **random-link** is nil, signal **implicit-error**.

If the **rank** of A or the **rank** of B is **greater-than one**, signal **rank-error**.

If either of A or B is neither a **scalar** nor a **one-element-vector**, signal **length-error**.

If either of A or B is not a **near-integer**, signal **domain-error**.

Set $A1$ to the **integer-nearest-to** A .

Set $B1$ to the **integer-nearest-to** B .

If either of $A1$ or $B1$ is not a **nonnegative-number**, signal **domain-error**.

If $A1$ is **greater-than** $B1$, signal **domain-error**.

If $A1$ is **zero**, return 1 0.

Using the **implementation-algorithm deal**, generate a **numeric vector** $Z0$ of **length** $A1$ whose elements are selected in a pseudorandom fashion without duplication from the **integers** in the **closed-interval-between zero** and $B1$ **minus one**, then set **random-link** to a new value.

Return $Z0$ **plus** **index-origin**.

Example:

```
12?300
2 3 42 94 70 105 215 9 110 298 201 5
```

Additional Requirements:

The selection of elements in Z is pseudorandom (see the operation **roll**). The elements of Z , their order, and the new value of **random-link** are determined by an **implementation-algorithm** whose only inputs are $A1$, $B1$ and **random-link**. The operation of **deal** is **atomic**: if **deal** signals an error, **random-link** shall be unchanged. The result of $A?B$, where A and B are **arrays**, shall be reproducible.

10.2.5 Compress

$$\begin{aligned} Z &\leftarrow A / B \\ Z &\leftarrow A \neq B \\ Z &\leftarrow A / [K] B \\ Z &\leftarrow A \neq [K] B \end{aligned}$$

Informal Description: Z is an array formed by selecting those subarrays, across a specified axis of B , for which the corresponding element of the **near-Boolean** vector or scalar A is tolerantly equal to **one** within **integer-tolerance**. Uses **index-origin**.

Evaluation Sequence:

If A is a **scalar**, set $A1$ to A .
 Otherwise, set $A1$ to A .
 If B is a **scalar**, set $B1$ to $(\rho A1) \rho B$.
 Otherwise, set $B1$ to B .

For form A/B

Return $A1/[\rho \rho B1] B1$ evaluated with **index-origin** set to **one**.

For form $A \neq B$

Return $A1/[1] B1$ evaluated with **index-origin** set to **one**.

For forms $A/[K] B$ and $A \neq [K] B$

If K is not a **valid-axis** for $B1$, signal **axis-error**.
 Otherwise, set $K1$ to the **integer-nearest-to** K .

If the **rank** of $A1$ is **greater-than one**, signal **rank-error**.

If the **count** of $A1$ is **one**, set $A2$ to $(\rho B1)[K1] \rho A1$.
 Otherwise, set $A2$ to $A1$.

If $\rho A2$ is not the same as $(\rho B1)[K1]$, signal **length-error**.

If any **item** of the **ravel-list** of $A2$ is not a **near-Boolean**, signal **domain-error**.

Set $A3$ to the **Boolean-array-nearest-to** $A2$.

If $B1$ is a **vector**, return $B1[(+/A3) \rho \nabla A3]$.

Otherwise, return an **array** Z such that

The **type** of Z is the same as the **type** of B ,

the **shape-list** of Z is $((K1 \neq 1 \rho \rho B1) \times \rho B1) + (K1 = 1 \rho \rho B1) \times +/A3$, evaluated with **comparison-tolerance** set to **zero**, and

the **ravel-list** of Z has the property that if $Z2$ and $B2$ are corresponding **vector-items** **along-axis** $K1$ of Z and $B1$ respectively, then $Z2$ is $A3/B2$.

Examples:

1 0 1 / 1 2 3
 1 3
 1 / 1 2 3
 1 2 3
 1 0 1 / 2
 2 2
 0 0 1 0 0 1 0 / [2] N2714
 1311 1312 1313 1314
 1611 1612 1613 1614
 2311 2312 2313 2314
 2611 2612 2613 2614
 p1/1
 1
 pp (,1)/2
 1

Note: Various error checks have been performed on *K* by the phrase evaluators, and *index-origin* is *one*, before this evaluation sequence is called.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

10.2.6 Expand

$$\begin{aligned} Z &\leftarrow A \setminus B \\ Z &\leftarrow A \setminus B \\ Z &\leftarrow A \setminus [K] B \\ Z &\leftarrow A \setminus [K] B \end{aligned}$$

Informal Description: Z is, for a **near-Boolean** scalar or vector A , an array of the same type as B containing subarrays of B along a specified axis.

For vector B , Z is such that $A/A \setminus B$ is B and, if P is the **typical-element** of B , $(\sim A)/A \setminus B$ is $(+/\sim A) \rho P$; similarly, $A/[K]A \setminus [K]B$ is B , $A \setminus A \setminus B$ is B , and $(\sim A)/[K]A \setminus [K]B$ and $(\sim A) \setminus A \setminus B$ are arrays of the **typical-element** of B . Uses **index-origin**.

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.

If any **item** of the **ravel-list** of A is not a **near-Boolean**, signal **domain-error**.

Set $A1$ to the **Boolean-array-nearest-to** A .

If B is a **scalar**, set $B1$ to $(+/\sim A) \rho B$.

Otherwise, set $B1$ to B .

For form $A \setminus B$

Return $A1 \setminus [\rho \rho B1] B1$ evaluated with **index-origin** set to **one**.

For form $A \setminus B$

Return $A1 \setminus [1] B1$ evaluated with **index-origin** set to **one**.

For forms $A \setminus [K] B$ and $A \setminus [K] B$

If K is not a **valid-axis** for $B1$, signal **axis-error**.

Otherwise, set $K1$ to the **integer-nearest-to** K .

If $(\rho B1)[K1]$ differs from $+/\sim A1$, signal **length-error**.

Return an **array** Z such that

the **type** of Z is the **type** of B ,

the **shape-list** of Z is $((K1 \neq \rho \rho B1) \times \rho B1) + (K1 = \rho \rho B1) \times \rho A1$, evaluated with **comparison-tolerance** set to **zero**, and

the **ravel-list** of Z has the property that $A1/[K1]Z$ is $B1$ and $(\sim A1)/[K1]Z$ is an **array** consisting entirely of the **typical-element** of B .

Examples:

```

1 0 1 \ 1 3
1 0 3
1 1 1 0 1 \ 'ABCD'
ABC D
1 0 1\ 2
2 0 2
1 0 1\[2] 2 2 p 'ABCD'
A B
C D
1 0 1 1\1 2 3
1 0 2 3
1 0 1 1\3
3 0 3 3
0 1 \ 3 1 p 3.14 2E17 -47
0 3.14E0
0 2.00E17
0 -4.70E1

1 0 1 0\[2] N224
111 112 113 114
0 0 0 0
121 122 123 124
0 0 0 0

211 212 213 214
0 0 0 0
221 222 223 224
0 0 0 0

```

Note: Various error checks have been performed on *K* by the phrase evaluators, and *index-origin* is one, before this evaluation sequence is called.

10.2.7 Rotate

$$\begin{aligned} Z &\leftarrow A \phi B \\ Z &\leftarrow A \ominus B \\ Z &\leftarrow A \phi[K] B \\ Z &\leftarrow A \ominus[K] B \end{aligned}$$

Informal Description: Z is an array of the same type and shape as B in which elements have been shifted cyclically along a specified axis. The amount and direction of shift is controlled by A . Uses **index-origin**.

Evaluation Sequence:

For form $A \phi B$

If B is a **scalar** and A is either a **scalar** or a **vector of length one**,

If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.
Otherwise, return B .

Otherwise, return $A \phi[\rho\rho B] B$ evaluated with **index-origin** set to **one**.

For form $A \ominus B$

If B is a **scalar** and A is either a **scalar** or a **vector of length one**,

If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.
Otherwise, return B .

Otherwise, return $A \phi[1] B$ evaluated with **index-origin** set to **one**.

For forms $A \phi[K] B$ and $A \ominus[K] B$

If K is not a **valid-axis** for B , signal **axis-error**.
Otherwise, set $K1$ to the **integer-nearest-to** K .

If A is a **scalar**, set $A1$ to $((K1 \neq 1 \rho\rho B) / \rho B) \rho A$, evaluated with **comparison-tolerance** set to **zero**.
Otherwise, set $A1$ to A .

If $A1$ is a **one-element-vector** and B is a **vector**, set $A2$ to $(10) \rho A1$.
Otherwise, set $A2$ to $A1$.

If the **rank** of B minus the **rank** of $A2$ is not **one**, signal **rank-error**.

If the **shape-list** of $A2$ is not the same as the **shape-list** of B with **axis** $K1$ omitted, signal **length-error**.

If any **item** of the **ravel-list** of $A2$ is not a **near-integer**, signal **domain-error**.
Set $A3$ to the **integer-array-nearest-to** $A2$.

If $A3$ is a **scalar** and B is a **vector**, return $B[1 + (\rho B) |^{-1} + A3 + 1 \rho B]$ evaluated with **comparison-tolerance** set to **zero**.

Otherwise return Z , an **array** such that the **shape-list** of Z is the same as the **shape-list** of B , the **type** of Z is the same as the **type** of B , and the **ravel-list** of Z has the property that if $Z0$ is a **vector-item along-axis** $K1$ of Z , $A0$ is the corresponding **item** of $A3$, and $B0$ is the corresponding **vector-item along-axis** $K1$ of B , then $Z0$ is $A0 \phi B0$.

Examples:

$3\phi 1\ 2\ 3\ 4\ 5$
 $4\ 5\ 1\ 2\ 3$
 $1\phi 1\ 2\ 3\ 4\ 5$
 $5\ 1\ 2\ 3\ 4$
 $7\phi 'ABCDEF'$
 $FABCDE$
 $1\theta N33$
 $21\ 22\ 23$
 $31\ 32\ 33$
 $11\ 12\ 13$
 $1\phi[1]N33$
 $21\ 22\ 23$
 $31\ 32\ 33$
 $11\ 12\ 13$
 $1\ 2\ 3\phi N34$
 $12\ 13\ 14\ 11$
 $23\ 24\ 21\ 22$
 $34\ 31\ 32\ 33$
 $N23\phi[2]N243$
 $141\ 112\ 123$
 $111\ 122\ 133$
 $121\ 132\ 143$
 $131\ 142\ 113$

 $221\ 232\ 243$
 $231\ 242\ 213$
 $241\ 212\ 223$
 $211\ 222\ 233$

Note: Various error checks have been performed on K by the phrase evaluators, and index-origin is one, before this evaluation sequence is called.

10.2.8 Base Value

$$Z \leftarrow A \perp B$$

Informal Description: Z is, for A and B numeric vectors, a number produced by regarding B as the representation of a number in the mixed radix number system specified by A .

If A or B is an array of rank greater than **one**, base value is like inner product: each **vector-item** along the last **axis** of A is applied to each **vector-item** along the first axis of B .

Evaluation Sequence:

Set $A0$ to $1 \rho \phi 1, \rho A$.

Set $B0$ to $1 \rho (\rho B), 1$.

If $A0$ differs from $B0$,

If A is a **scalar** or **one-element-vector**, return $(B0 \rho A) \perp B$.

If B is a **scalar** or **one-element-vector**, return $A \perp A0 \rho B$.

Otherwise, signal **length-error**.

If $A0$ is the same as $B0$,

If any **item** of the **ravel-list** of A is not a **number**, signal **domain-error**.

If any **item** of the **ravel-list** of B is not a **number**, signal **domain-error**.

Return $(\phi \phi (\phi \rho A) \rho \phi \times \backslash \phi A, 1) + . \times B$.

Examples:

```

10 1 1 2 3
123
24 60 60 1 1 2 3
3723
  A ← 2 3 10 10 10 12 60 60
10 10 10
12 60 60
  B ← 3 2 1 4 2 5 3 6
1 4
2 5
3 6
  A ⊥ B
123 456
3723 14706
  .001 10 10 1 1 2 3
123
60 1 1 2 3
3723
  '1 13
0
  'A' 1 10
0
```

Note: The shape requirements of A and B are intentionally stricter in this definition than in several existing systems. Specifically, this standard does not require that if the last **axis** of A is **one** it be replicated to match the first **axis** of B , or that if the first **axis** of B is **one** it be replicated to match the last **axis** of A .

The first element of each left argument has no actual effect on the result of base value, but permits use of the same left argument for base value and representation. If A is a positive numeric vector and B a non-negative numeric scalar such that $B < \times / A$, then $A \perp A \top B$ is B .

10.2.9 Representation

$$Z \leftarrow A \uparrow B$$

Informal Description: Z is the representation of B in mixed radix number system A .

Evaluation Sequence:

- If A or B is **empty**, return $((pA), pB)p0$.
- If any **item** of the **ravel-list** of A is a **character**, signal **domain-error**.
- If any **item** of the **ravel-list** of B is a **character**, signal **domain-error**.
- If A is **scalar**, return $A \mid B$, evaluated with **comparison-tolerance** set to **zero**.
- If A is a **vector** and B is a **scalar**,

Generate two **numeric vectors** Z and C that satisfy the following constraints:

- The **length** of Z is pA .
- The **length** of C is $1+pA$.
- $C[1+pA]$ is B .
- For all **scalar** indices I in $1:pA$:

- $Z[I]$ is $A[I] \uparrow C[I+1]$.
- If $A[I]$ is **zero**, $C[I]$ is **zero**;
- Otherwise, $C[I]$ is $(C[I+1] - Z[I]) \div A[I]$.

Return Z .

Otherwise, return $Z1$, an **array** such that the **type** of $Z1$ is **numeric**, the **shape-list** of $Z1$ is $(pA), pB$ and the **ravel-list** of $Z1$ has the following property:

- Let I stand for an **item** of the **index-set** of the **ravel-along-axis one** of A .
- Let $A1$ stand for **vector-item** I of the **ravel-along-axis one** of A .
- Let J stand for an **item** of the **index-set** of the **ravel-list** of B .
- Let $B1$ stand for **item** J of the **ravel-list** of B .
- Let N stand for the **count** of B .
- Let P stand for $J + (N \times (I - 1))$.

Then, **vector-item** P of the **ravel-list** of $Z1$ **along-axis one** is $A1 \uparrow B1$.

Examples:

```

      10 10 10τ123
1 2 3      10 10 10τ123 456
1 4
2 5
3 6
      A←11 3p10 16 2
      '0123456789ABCDEF'[q 1+Aτ1000 1024]
00000001000
000000003E8
01111101000

00000001024
00000000400
10000000000
      2 2 2 τ -1
1 1 1      0 2 2 τ -1
-1 1 1      0 1 τ 3.75 -3.75
3      -4
0.75 0.25

```

Note: The shape of the result of *representation* is always $(pA), pB$.

The *system-parameter comparison-tolerance* is not an implicit argument of *representation*.

10.2.10 Dyadic Transpose

$$Z \leftarrow A \text{ } \text{ } B$$

Informal Description: Z is an array formed by rearranging and possibly coalescing the axes of B according to the vector A . Each element of A corresponds to an axis of B by position and to an axis of Z by value. The largest value in A determines the rank of Z . All axes of Z must be present in A . If A contains no repeated elements, the shape of Z is $(\rho B)[\Delta A]$. Repeated elements in A select diagonals from B . The length of the corresponding axis of Z is the shortest of the lengths of the designated axes of B . Uses **index-origin**.

Evaluation Sequence:

If **index-origin** is **nil**, signal **implicit-error**.

If A is a **scalar**, set $A1$ to $,A$.

Otherwise, set $A1$ to A .

If $A1$ is not a **vector**, signal **rank-error**.

If the **length** of $A1$ is not the same as the **rank** of B , signal **length-error**.

If any **item** of the **ravel-list** of $A1$ is not a **near-integer**, signal **domain-error**.

Set $A2$ to the **integer-array-nearest-to** $A1$.

Generate $A3$, a **numeric array** with the **shape-list** of $A2$ such that each **item** of the **ravel-list** of $A3$ is **(one minus index-origin)** plus the corresponding element of $A2$.

If $\wedge / A3 \in \uparrow / 0, A3$ evaluated with **comparison-tolerance** set to **zero** is not **one**, signal **domain-error**.

If $\wedge / (\uparrow / 0, A3) \in A3$ evaluated with **comparison-tolerance** set to **zero** is not **one**, signal **domain-error**.

Return an **array** Z having the following properties:

The **rank** of Z is $\uparrow / 0, A3$.

The **type** of Z is the same as the **type** of B .

The **shape-list** of Z is such that for all I in the **index-set** of the **shape-list** of Z , **item** I of the **shape-list** of Z is $\uparrow / (A3 = I) / \rho B$, evaluated with **comparison-tolerance** set to **zero**.

The **ravel-list** of Z is such that, for all J in the **index-set** of the **ravel-list** of Z , **item** J of the **ravel-list** of Z is **item** $1 + (\rho B) \downarrow ((\rho Z) \uparrow J - 1) [A3]$ of the **ravel-list** of B .

Examples:

```

      1 3 2 4
111 121 131
112 122 132
113 123 133
114 124 134

211 221 231
212 222 232
213 223 233
214 224 234
      1 1 3 4
11 22 33

```

3 1 2 3 4

111 211
112 212
113 213
114 214

121 221
122 222
123 223
124 224

131 231
132 232
133 233
134 234

pp(10)5
0

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

10.2.11 Take

$$Z \leftarrow A \uparrow B$$

Informal Description: Z is an array having shape ρA . Informally, Z is a corner of B . The choice of corner is based on the signs of the elements of A . If the absolute values of elements of A are greater than corresponding elements of ρB , Z is padded with the **typical-element** of B .

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.
Set $A1$ to ρA .

If B is a **scalar**, set $B1$ to $((\rho A1) \rho 1) \rho B$.
Otherwise, set $B1$ to B .

If the **shape-list** of $A1$ does not match the **rank** of $B1$, signal **length-error**.
If any **item** of the **ravel-list** of $A1$ is not a **near-integer**, signal **domain-error**.
Set $A2$ to the **integer-array-nearest-to** $A1$.

Return an **array** Z such that the **type** of Z is the same as the **type** of B , the **shape-list** of Z is $\rho A2$, and the **ravel-list** of Z has the property that for each **scalar** I in the **index-set** of the **ravel-list** of Z , **item** I of the **ravel-list** of Z is determined as follows:

Let J stand for $((\rho A2) \uparrow I - 1) + (A2 < 0) \times A2 + \rho B1$ evaluated with **comparison-tolerance** set to **zero**.

If each element of J is in the **open-interval-between negative-one** and the corresponding element of $\rho B1$, **item** I of the **ravel-list** of Z is $(\rho B1)[1 + (\rho B1) \downarrow J]$.
Otherwise, **item** I of the **ravel-list** of Z is the **typical-element** of B .

Examples:

```

      2 ↑ N5
1 2   -2 ↑ N5
4 5   -2 6 ↑ N44
31 32 33 34 0 0
41 42 43 44 0 0
      -4 -4 ↑ 99
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 99

```

10.2.12 Drop

$$Z \leftarrow A \downarrow B$$

Informal Description: Z is a corner of B with shape $0 \uparrow (\rho B) - |A$. The choice of corner is based on the signs of the elements of A .

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.

Set $A1$ to A .

If B is a **scalar**, set $B1$ to $((\rho A1) \rho 1) \rho B$.

Otherwise, set $B1$ to B .

If the **shape-list** of $A1$ does not match the **rank** of $B1$, signal **length-error**.

If any **item** of the **ravel-list** of $A1$ is not a **near-integer**, **near-integer**, signal **domain-error**.

Set $A2$ to the **integer-array-nearest-to** $A1$.

Return $(((A2 < 0) \times 0 \uparrow A2 + \rho B1) + (A2 \geq 0) \times 0 \downarrow A2 - \rho B1) \uparrow B1$ evaluated with **comparison-tolerance** set to **zero**.

Examples:

```

      1 ↓ N5
2 3 4 5
      2 -1 ↓ N44
31 32 33
41 42 43
      ρ1 ↓ 5
0
      ρ0 ↓ 5
1
      ρ1 2 3 ↓ 4
0 0 0
      ' ' ↓ 5
5
      ρρ ' ' ↓ 5
0
```

10.2.13 Matrix Divide

$$Z \leftarrow A \oslash B$$

Informal Description: Z is the least squares solution to a linear system specified by A and B .

Evaluation Sequence:

If the **rank** of A or the **rank** of B is **greater-than two**, signal **rank-error**.

Set $A1$ to $(2 \uparrow (\rho A), 1 \ 1) \rho A$.

Set $B1$ to $(2 \uparrow (\rho B), 1 \ 1) \rho B$.

If $1 \uparrow \rho A1$ is not the same as $1 \uparrow \rho B1$, signal **length-error**.

If $1 \uparrow \rho B1$ is **less-than** $1 \uparrow \rho A1$, signal **length-error**.

If any **item** of the **ravel-list** of A is not a **number**, signal **domain-error**.

If any **item** of the **ravel-list** of B is not a **number**, signal **domain-error**.

Use the **implementation-algorithm matrix-divide** to generate an **array** Z , such that the **type** of Z is **numeric**, the **shape-list** of Z is $(1 \uparrow \rho B1), 1 \uparrow \rho A1$, and the **ravel-list** of Z has the property that for each **scalar** I in $1 \uparrow \rho A1$, $Z[; I]$ is such that it minimises $+(A1[; I] - B1+. \times Z[; I])^2$.

If Z cannot be uniquely determined within a reasonable round-off criterion by the above constraints, signal **domain-error**.

Return $((1 \uparrow \rho B), 1 \uparrow \rho A) \rho Z$.

Additional Requirements:

The round-off criterion discussed above is not specified by this standard.

Note: The following article, Martin, G. A., *The Solutions of Linear Systems in APL: Towards An Extension of Matrix Divide APL80 Noordwijkerhout June 24-26, 1980* — Published by Gijsbert van der Linden, North-Holland Publishing Company Amsterdam. New York. Oxford, describes an acceptable algorithm for matrix division.

The following document also deals with this subject: Jenkins, M. A., *Domino — An APL Primitive Function for Matrix Inversion — Its Implementation and Applications. APL Quote-Quad Vol III No. 4, February 1972, pp. 4-15.*

10.2.14 Indexed Reference

$$Z \leftarrow A[I]$$

Informal Description: Z is an array consisting of elements of the array A selected and structured by the position and values of the arrays in the **index-list** I . Uses **index-origin**.

Evaluation Sequence:

If the **number-of-items** in the **index-list** I is not the same as the **rank** of A , signal **rank-error**.
In the following,

Let IX stand for (array) item X of I .
Let JX stand for (array) item X of J .
Let LX stand for (array) item X of L .
Let LY stand for (array) item $X+1$ of L .

Generate J , a list of arrays having the count of I , such that for every member X of the **index-set** of I ,

If IX is an **elided-index-marker**, set JX to $^{-1}+1(pA)[X]$.
Otherwise,

If any item of the **ravel-list** of IX is not a **near-integer**, signal **domain-error**.
Set JX to the **integer-array-nearest-to** IX .
Set JX to JX minus **index-origin**.
If any element of JX is not an element of $^{-1}+1(pA)[X]$, signal **index-error**.

Set JX to $JX \times x / X + pA$.

Generate L , a list of arrays with $1+pA$ (array) items such that

Item $1+pA$ of L is **one**.
For all X in $1+pA$, LX is $JX \circ . + LY$.

Let K stand for **first-item** in the list of arrays L .
Return $(,A)[K]$.

Examples:

```

1 2 3[2]
2
N222[2 1;;2]
212 222
112 122

```

Note: Since an **index-list** can never have zero items, indexing will always signal **rank-error** when argument A is a scalar.

The vector indexing on which this subsection is based is defined in the subsection **evaluate-indexed-reference**.

10.2.15 Indexed Assignment

$$Z \leftarrow V[I] \leftarrow B$$

Informal Description: Z is B . As a side effect, indexed assignment sets elements of the array V selected by the position and values of the arrays in the **index-list** I to corresponding elements of B . Note that, at this stage, V is known to be a **variable-name**, not a **value**. Uses **index-origin**.

Evaluation Sequence:

If **index-origin** is nil, signal **implicit-error**.

Set A to the **current-content** of V .

If the **number-of-items** in the **index-list** I does not match the **rank** of A , signal **rank-error**.

In the following,

Let IX stand for (array) **item** X of I .

Let JX stand for (array) **item** X of J .

Let LX stand for (array) **item** X of L .

Let LY stand for (array) **item** $X+1$ of L .

Generate J , an **index-list** having the same **count** as I such that for every **item** X of the **index-set** of I ,

If IX is an **elided-index-marker**, set JX to $^{-1}+1(\rho A)[X]$.

Otherwise,

If any **item** of the **ravel-list** of IX is not a **near-integer**, signal **domain-error**.

Set JX to the **integer-array-nearest-to** IX .

Set JX to JX **minus index-origin**.

If any element of JX is not an element of $^{-1}+1(\rho A)[X]$, signal **index-error**.

Set JX to $JX \times \times / X \rho A$.

Generate L , an **index-list** with $1+\rho \rho A$ (array) **items** such that

Item $1+\rho \rho A$ of L is **one**.

For all X in $1+\rho \rho A$, LX is $JX \circ . + LY$.

Let K stand for **first-item** in the **list of arrays** L .

Set $K1$ to $((1 \mp \rho K) / \rho K) \rho K$ evaluated with **comparison-tolerance** set to **zero**.

Set $B1$ to $((1 \mp \rho B) / \rho B) \rho B$ evaluated with **comparison-tolerance** set to **zero**.

If $B1$ is a **scalar**, set $B2$ to $(\rho K1) \rho B1$.

Otherwise, set $B2$ to $B1$.

If the **rank** of $K1$ is not the same as the **rank** of $B2$, signal **rank-error**.

If any **item** of the **shape-list** of $K1$ is not the same as the corresponding **item** of the **shape-list** of $B2$, signal **length-error**.

If $K1$ is not **empty** and the **type** of A differs from the **type** of B , signal **domain-error**.

Set M to **one**.

Repeat:

If M is not **greater-than** the **count** of $K1$,

Let $K2$ stand for **item** M of the **ravel-list** of $K1$.

Set **item** $K2$ of the **ravel-list** of A to **item** M of the **ravel-list** of B .

Set M to M **plus one**.

Otherwise,

Set the **current-referent** of V to a token whose **class** is **variable** and whose **content** is A .

Return a **token** whose **class** is **committed-value** and whose **content** is B .

(End of repeated block)

Examples:

```

      X ← 1 2 3
      □ ← X[3 2] ← 4 5
4 5
      X
1 5 4
      □ ← Y ← N222
111 112
121 122

211 212
221 222
      □ ← Y[2 1;;1] ← N12121
11111
11121

12111
12121
      Y
12111 112
12121 122

11111 212
11121 222

```

Additional Requirements:

The evaluation sequence requires that the value of $A[1]$ after the statement $A[1 1 1] ← 1 2 3$ has been evaluated be 3.

Indexed assignment exhibits **atomic** behaviour. If **indexed assignment** signals an error, the value of V shall be unchanged.

Note: Since an **index-list** can never have zero items, indexing will always signal **rank-error** when argument A is a scalar.

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

11 SYSTEM FUNCTIONS

11.1 INTRODUCTION

Note: System functions are primitive functions whose names are *distinguished-identifiers* rather than *primitives*. System function names are not permitted in the *locals-list* of a *defined-function header-line*.

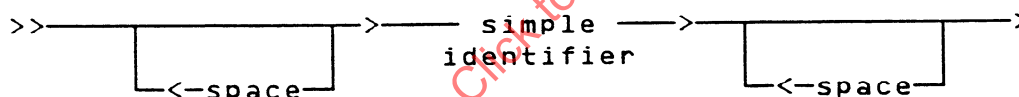
The system functions related to **shared-variables** and **defined-functions** are specified in those chapters.

11.2 DEFINITION

Note: Some system functions take an argument which is treated as an identifier-array.

Identifier-Array: A character array of rank two each of whose rows matches the character-diagram identifier-row.

11.3 DIAGRAM

Identifier-Row

11.4 NILADIC SYSTEM FUNCTIONS

Note: A niladic system function is an implementation-provided facility associated with a distinguished-identifier. The primary difference between a niladic system function and a system variable is that a niladic system function causes an error to be signalled if its name is included in the list of locals of a defined function header.

11.4.1 Time Stamp

$Z \leftarrow \square TS$

Informal Description: Z is a seven-element numeric vector representing the current date and time relative to an epoch and a time zone. The choice of epoch and time zone are not specified by this standard.

Evaluation Sequence:

Return the seven-element **numeric vector** result of the **implementation-algorithm time-stamp**.

Example:

$\square TS$
1789 7 14 11 14 45 586.4

Additional Requirements:

The first through sixth elements of the result returned by **time-stamp** are integral and represent respectively the current year, month, day, hour, minute, and second relative to the underlying epoch and time zone. The hour corresponds to the number of integral hours that have elapsed within the current day.

The seventh and last element is a quantity, not necessarily integral, representing the milliseconds that have elapsed within the current second.

The accuracy, resolution, epoch, and time zone for **time-stamp** are not specified by this standard.

11.4.2 Atomic Vector

$$Z \leftarrow \square AV$$

Informal Description: Z is an **implementation-defined character vector** containing every element of the **character-set** exactly once.

Evaluation Sequence:

Return **atomic-vector**.

*Note: The elements of the **atomic-vector** are often ordered so that, in **index-origin zero**,*

$((\lceil 2 \otimes \square AV) \rho 2) \tau \square AV \uparrow 'X'$ gives the bit representation of $'X'$. This behaviour is not required.

11.4.3 Line Counter

$$Z \leftarrow \square LC$$

Informal Description: Z is a vector of statement numbers of active functions ordered so that the most recently called function has the lowest index.

Evaluation Sequence:

Set Y to a **numeric vector** whose **length** is the same as the **number-of-items** in the **state-indicator**, such that for every I in $1 \rho Y$, $Y[I]$ is the **current-line-number** of item I of the **state-indicator**, if the **mode** of item I of the **state-indicator** is **defined-function**, and **zero** otherwise.

Set M to a **Boolean vector** whose **length** is the same as the **number-of-items** in the **state-indicator**, such that for every I in $1 \rho M$, $M[I]$ is **one** if the **mode** of item I of the **state-indicator** is **defined-function**, and **zero** otherwise.

Return M/Y .

*Note: The operations that affect the value of $\square LC$ are discussed in the subsection **defined-function-control**.*

11.5 MONADIC SYSTEM FUNCTIONS

Note: Refer to the *Defined Functions* chapter for definitions of $\square FX$ and $\square CR$.

11.5.1 Delay

$Z \leftarrow \square DL \ B$

Informal Description: Z is the time, in seconds, for completion of this operation. As a side effect, $\square DL$ causes a delay in execution of at least B seconds.

Evaluation Sequence:

Set $T0$ to **current-time** seconds.

If B is not a **scalar**, signal **rank-error**.

If B is not a **number**, signal **domain-error**.

Set $T1$ to $T0$ **plus** B .

Wait until **current-time** is not **less-than** $T1$.

Return Z , a **numeric scalar** such that the **first-item** in the **ravel-list** of Z is **current-time** minus $T0$.

Note: B may be fractional or negative.

11.5.2 Name Class

$$Z \leftarrow \square NC \ B$$

Informal Description: Z is a vector of name classes giving the usage of the identifier in the corresponding row of B . 0 means an identifier is available without current referent; 1 a label; 2, a variable or a shared variable; 3, a defined function.

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

If B is **empty**, return 1 0.

Set $B1$ to $(\neg 2 \uparrow 1 \ 1, \rho B) \rho B$.

If any row of $B1$ does not match **identifier-row**, signal **domain-error**.

Generate Z , a **numeric vector** such that the **length** of Z is the same as the **number-of-rows** in $B1$, and the **ravel-list** of Z has the property that for every I in $1 1 \uparrow \rho B$, the following holds:

Let ZI stand for **item** I of the **ravel-list** of Z .

Let N stand for the **token** that matched the **character-diagram simple-identifier** in the **identifier-row**.

If the **current-class** of N is **nil**, ZI is **zero**.

If the **current-class** of N is **label**, ZI is **one**.

If the **current-class** of N is **variable** or **shared-variable**, ZI is **two**.

If the **current-class** of N is **defined-function** or **niladic-defined-function**, ZI is **three**.

Otherwise, signal **domain-error**.

Return Z .

Example:

```

      V Z←TEST;R0;R21;R22;R31;R32
[1]  R1:R21←1  SVO 'R22'
[2]  Z←FX 1 3ρ'R31'
[3]  Z←FX 1 5ρ'R32 X'
[4]  Z←NC 6 3ρ'R0 R1 R21R22R31R32'
      V
      TEST
0 1 2 2 3 3

```

11.5.3 Expunge

$$Z \leftarrow \square EX B$$

Informal Description: Z is a Boolean vector such that an element of Z is **one** if the identifier in the corresponding row of B is available for use when the operation completes. Expunge changes the **current-referent** of **symbols** whose **current-class** is **variable**, **shared-variable**, **defined-function**, or **niladic-defined-function** to **nil**, making them available for redefinition. The **symbols** to be changed are named by the rows of B , an **identifier-array**.

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Set $B1$ to $(\neg 2 \uparrow 1 \ 1, pB) pB$.

For each **row** J of $B1$,

If **row** J of $B1$ does not match **identifier-row**, signal **domain-error**.

Let N stand for the **token** that matched the **character-diagram simple-identifier** in **identifier-row**.

If the **current-class** of N is **shared-variable**, retract N .

If the **current-class** of N is **variable**, set the **current-referent** of N to a **token** whose **class** is **nil**.

If the **current-class** of N is **defined-function** or **niladic-defined-function**, and if the **current-referent** of N is **locally-erasable**, set the **current-referent** of N to a **token** whose **class** is **nil**.

Return $\sim \times \square NC B$.

Additional Requirement:

The operation of Expunge is **atomic**: if Expunge signals an error, **current-referents** shall be unchanged.

*Note: Because of the definition of **locally-erasable**, a **defined-function** cannot be expunged by a **conforming-program** while it is **pendent** or **waiting**. This does not prevent a **conforming-implementation** from permitting such behaviour; it simply leaves the consequences to a **conforming-program** undefined until some future standard.*

11.5.4 Name List

$$Z \leftarrow \square NL B$$

Informal Description: Z is an **identifier-array**. An identifier occurs in Z if its **name-class** is in B . Elements of B can be: 1, for labels; 2, for variables or shared variables; 3, for user defined functions.

Evaluation Sequence:

If the **rank** of B is **greater-than one**, signal **rank-error**.

If any **item** of the **ravel-list** of B is not a **near-integer**, signal **domain-error**.

Set $B1$ to the **integer-array-nearest-to** B .

If $B1$ contains any **numbers** other than **one**, **two**, or **three**, signal **domain-error**.

Generate Z , an **identifier-array** consisting of the **names** of **symbols** whose **current-class** is **label**, **variable**, **shared-variable**, **niladic-defined-function**, or **defined-function**. The **symbol names** are left-justified in the rows of Z , and Z has no all-blank trailing columns.

Return $((\square NC Z) \in B1) \neq Z$.

*Note: The **rank** of Z is always **two**. If there are no **symbols** with a **name-class** designated in B , Z is $0 \ 0 p \ ' \ '$.*

11.5.5 Query Stop

$Z \leftarrow \square STOP B$

Informal Description: Z is an integer vector of line numbers in the function whose name is B that have stop control set.

Evaluation Sequence:

If the rank of B is **greater-than one**, signal **rank-error**.

If B does not match **identifier-row**, signal **domain-error**.

Let N stand for the **token** that matched the **character-diagram simple-identifier** in **identifier-row**.

If the **current-class** of N is not **defined-function** or **niladic-defined-function**, signal **domain-error**.

Return the **stop-vector** of the **current-content** of N .

Additional Requirement:

Query Stop is part of the **optional-facility Trace-and-Stop-Control**.

Note: Query Stop is not origin sensitive.

If no lines have stop control set, the result is 1 0.

11.5.6 Query Trace

$Z \leftarrow \square TRACE B$

Informal Description: Z is an integer vector of line numbers in the function whose name is B that are being traced.

Evaluation Sequence:

If the rank of B is **greater-than one**, signal **rank-error**.

If B does not match **identifier-row**, signal **domain-error**.

Let N stand for the **token** that matched the **character-diagram simple-identifier** in **identifier-row**.

If the **current-class** of N is not **defined-function** or **niladic-defined-function**, signal **domain-error**.

Return the **trace-vector** of the **current-content** of N .

Additional Requirement:

Query Trace is part of the **optional-facility Trace-and-Stop-Control**.

Note: Query Trace is not origin sensitive.

If no lines are being traced, the result is 1 0.

11.6 DYADIC SYSTEM FUNCTIONS

11.6.1 Name List

$Z \leftarrow A \square NL B$

Informal Description: Z is an **identifier-array** of all names having a **name-class** in B that begin with a letter in A .

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.
 If any **item** of the **ravel-list** of A does not match **letter**, signal **domain-error**.
 Set $Z1$ to $\square NL B$.
 If $0 \in p Z1$, return $Z1$.
 Return $(Z1[:1] \in A) \neq Z1$.

11.6.2 Set Stop

$Z \leftarrow A \square STOP B$

Informal Description: Z is a numeric vector representing the stop controls in effect for the function whose name is in B before $\square STOP$ began evaluation. As a side effect, $\square STOP$ sets stop controls for the function whose name is in B at lines specified by A .

Evaluation Sequence:

If the **rank** of B is **greater-than one**, signal **rank-error**.
 If B does not match **identifier-row**, signal **domain-error**.
 Let N stand for the **token** that matched the **character-diagram simple-identifier** in **identifier-row**.
 If the **current-class** of N is not **defined-function** or **niladic-defined-function**, signal **domain-error**.

 If the **rank** of A is **greater-than one**, signal **rank-error**.
 If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.
 Set $A1$ to the **integer-array-nearest-to** A .
 If any **item** of the **ravel-list** of $A1$ is not a **positive-integer**, signal **domain-error**.
 Set Z to $\square STOP B$.
 Let M stand for the **last-line-number** of the **current-content** of N .
 Set the **stop-vector** of the **current-content** of N to $((1M) \in A1) / 1M$, evaluated with **index-origin one**.
 Return Z .

Additional Requirement:

Set Stop is part of the **optional-facility Trace-and-Stop-Control**.

Note: Stop controls are an attribute of **defined-function** objects. $\square STOP$ affects the **current-referent** of B only.

11.6.3 Set Trace

$Z \leftarrow A \square TRACE B$

Informal Description: Z is a numeric vector representing the lines being traced in the function whose name is in B before $\square TRACE$ began evaluation. As a side effect, $\square TRACE$ begins tracing the lines specified by A in the function whose name is in B .

Evaluation Sequence:

If the **rank** of B is **greater-than one**, signal **rank-error**.
 If B does not match **identifier-row**, signal **domain-error**.
 Let N stand for the **token** that matched the **character-diagram simple-identifier** in **identifier-row**.
 If the **current-class** of N is not **defined-function** or **niladic-defined-function**, signal **domain-error**.

 If the **rank** of A is **greater-than one**, signal **rank-error**.
 If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.
 Set $A1$ to the **integer-array-nearest-to** A .
 If any **item** of the **ravel-list** of $A1$ is not a **positive-integer**, signal **domain-error**.
 Set Z to $\square TRACE B$.
 Let M stand for the **last-line-number** of the **current-content** of N .
 Set the **trace-vector** of the **current-content** of N to $((1M) \in A1) / 1M$, evaluated with **index-origin one**.
 Return Z .

Additional Requirement:

Set Trace is part of the **optional-facility Trace-and-Stop-Control**.

Note: Trace controls are an attribute of **defined-function** objects. $\square TRACE$ affects the **current-referent** of B only.

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

12 SYSTEM VARIABLES

12.1 DEFINITIONS

System-Variable-Symbol: A **symbol**, having as its name a **distinguished-identifier**, and as its **referent-list** a list of **tokens** representing the value assigned to an associated **system-parameter** in each **context** of the **state-indicator**.

Note: System parameters are values used implicitly and set as side effects by primitive operations.

Internal-Value-Set: An **implementation-defined** set of values that the **tokens** in the **referent-list** of a **system-variable-symbol** can take on.

Note: When a **system-variable-symbol** is **localised** the initial **class** of its associated **system-parameter** is **nil**. If a primitive operation is invoked that requires a system parameter whose class is currently **nil**, the primitive operation signals **implicit-error**. Therefore, a **conforming-program** that **localises** **system-variable-symbols** should assign them values from their **internal-value-set** before calling primitive operations that require them.

The assignment operation for a system variable rejects attempts to set its associated system parameter to a value outside its **internal-value-set**.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

12.2 EVALUATION SEQUENCES

12.2.1 Comparison Tolerance

$Z \leftarrow \square CT \leftarrow B$
 $Z \leftarrow \square CT$

Informal Description: Z is the current value of **comparison-tolerance** .

The distance within which numbers are to be considered equal by certain primitives is a function of **comparison-tolerance** .

Evaluation Sequence:

For form $\square CT$

Return **comparison-tolerance** .

For form $\square CT \leftarrow B$

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is not **one**, signal **length-error**.

Let $B1$ be the **first-scalar** in B .

If $B1$ is not a **nonnegative-number**, signal **domain-error**.

If $B1$ is not in the **internal-value-set** of **comparison-tolerance** , signal **limit-error**.

Set **comparison-tolerance** to $B1$.

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Additional Requirements:

The **internal-value-set** for **comparison-tolerance** consists of **nonnegative-numbers** not greater than the **implementation-parameter comparison-tolerance-limit**.

The initial value of **comparison-tolerance** in a **clear-workspace** is that member of the **internal-value-set** for **comparison-tolerance** given by the **implementation-parameter initial-comparison-tolerance**.

*Note: The following subsections reference **comparison-tolerance** : **ceiling**, **equal**, **floor**, **greater than or equal to**, **greater than**, **index of**, **less than or equal to**, **less than**, **member of**, **not equal**, and **residue**.*

12.2.2 Random Link

$Z \leftarrow \square RL \leftarrow B$
 $Z \leftarrow \square RL$

Informal Description: Z is the current value of **random-link** ; **random-link** is the current seed of the pseudorandom number generator.

Evaluation Sequence:

For form $\square RL$

Return **random-link** .

For form $\square RL \leftarrow B$

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is not **one**, signal **length-error**.

Set $B1$ to the **first-scalar** in B .

If $B1$ is not a **near-integer**, signal **domain-error**.

Set $B2$ to the **integer-nearest-to** $B1$.

If $B2$ is **less-than one**, signal **domain-error**.

If $B2$ is not in the **internal-value-set** for **random-link** , signal **limit-error**.

Set **random-link** to $B2$.

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Additional Requirements:

The **internal-value-set** of **random-link** is **implementation-defined**.

The initial value of **random-link** in a **clear-workspace** is that member of the **internal-value-set** for **random-link** given by the **implementation-parameter** **initial-random-link**.

Note: The system parameter **random-link** is used and set by **roll** and **deal**. **Roll** gives a reference for a suitable algorithm.

12.2.3 Print Precision

$Z \leftarrow \square PP \leftarrow B$
 $Z \leftarrow \square PP$

Informal Description: Z is the current value of **print-precision**, which controls the number of significant positions in the output form produced by **monadic format** and **numeric-output-conversion**.

Evaluation Sequence:

For form $\square PP$

Return **print-precision**.

For form $\square PP \leftarrow B$

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is not **one**, signal **length-error**.

If B is not **near-integer**, signal **domain-error**.

Set $B1$ to the **integer-nearest-to** the **first-scalar** in B .

If $B1$ is **less-than one**, signal **domain-error**.

If $B1$ is not in the **internal-value-set** of **print-precision**, signal **limit-error**.

Set **print-precision** to $B1$.

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Additional Requirements:

The **internal-value-set** of **print-precision** consists of the **integer** scalars in the **closed-interval-between one** and **print-precision-limit**.

The initial value of **print-precision** in a **clear-workspace** is that member of the **internal-value-set** for **print-precision** given by the **implementation-parameter** **initial-print-precision**.

Note: The system parameter **print-precision** is used by **monadic format** and **numeric-output-conversion**. **Print-precision-limit** should be greater than or equal to **full-print-precision**.

12.2.4 Index Origin

$$Z \leftarrow \square IO \leftarrow B$$

$$Z \leftarrow \square IO$$

Informal Description: Z is the current value of **index-origin**, which is the index associated with the first position of any axis of non-zero length.

Evaluation Sequence:

For form $\square IO$

Return **index-origin**.

For form $\square IO \leftarrow B$

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is not **one**, signal **length-error**.

If B is not a **near-integer**, signal **domain-error**.

Set $B1$ to the **integer-nearest-to** the **first-scalar** in B .

If $B1$ is not in the **internal-value-set** of **index-origin**, signal **limit-error**.

Set **index-origin** to $B1$.

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Additional Requirements:

The **internal-value-set** for **index-origin** is the **scalar integer** values **zero** and **one**.

The initial value of **index-origin** in a **clear-workspace** is that member of the **internal-value-set** for **index-origin** given by the **implementation-parameter** **initial-index-origin**.

Note: The following primitive operations refer to **index-origin**: **deal**, **dyadic transpose**, **grade down**, **grade up**, **index generator**, **index of**, **indexed assignment**, **indexed reference**, **roll**, and all functions that provide for axis specification when the form containing an axis is used.

12.2.5 Latent Expression

$Z \leftarrow \square LX \leftarrow B$
 $Z \leftarrow \square LX$
 $Z \leftarrow \square LX[I] \leftarrow B$

Informal Description: Z is the current value of **latent-expression**, which is a **character** vector that is executed when a workspace is activated.

Evaluation Sequence:

For form $\square LX$

Return **latent-expression**.

For form $\square LX \leftarrow B$

If the **rank** of B is **greater-than one**, signal **rank-error**.

If any **item** of the **ravel-list** of B is not a **character**, signal **domain-error**.

If B is not in the **internal-value-set** of **latent-expression**, signal **limit-error**.

Set **latent-expression** to B .

Return a **token** whose **class** is **committed-value** and whose **content** is B .

For form $\square LX[I] \leftarrow B$

Set LX to **latent-expression**.

Evaluate $LX[I] \leftarrow B$.

Set **latent-expression** to LX .

Return a **token** whose **class** is **committed-value** and whose **content** is B .

Additional Requirements:

The **internal-value-set** for **latent-expression** is an **implementation-defined** subset of the set of all character vectors which includes the empty vector.

The initial value of **latent-expression** in a **clear-workspace** is that member of the **internal-value-set** for **latent-expression** given by the **implementation-parameter** **initial-latent-expression**.

13 DEFINED FUNCTIONS

13.1 INTRODUCTION

Note: Algorithms written in APL are called defined functions. A defined function consists of a header line and zero or more body lines. The header line indicates the name and syntax class of the function and gives a list of names to be localised and, in the case of argument names, initialised. Each body line consists of an optional label followed by an APL line to be evaluated. Body lines are evaluated in the order in which they occur in the defined function unless a statement beginning with a right arrow is evaluated.

A defined function is established in a workspace by the system function `⌈FX`, by actions in function definition mode, or by the system command `)COPY`.

In this standard, a **defined-function** is represented as an object with three attributes: **canonical-representation**, **trace-vector**, and **stop-vector**. A **defined-function** is called when its name occurs in a prefix of the **current-stack** matching one of the patterns in the **phrase-table**.

A **defined-function** is called by the operation **call-defined-function**. This operation prefixes the **state-indicator** with a new context, localises any local names in the header-line, and calls the evaluation sequence in the subsection **defined-function-control**.

A **defined-function** ends the evaluation if **defined-function-control** finds **current-line-number** set to a value not in the **closed-interval-between one and last-line-number**. At this point, the first item of the **state-indicator** is discarded and a token of class **result** is returned, via **call-defined-function**, to the **phrase-evaluator** that called it.

The returned token may be **constant** (for functions returning a value), **nil** (for functions that return no value), **unwind** (for functions that end through the evaluation of an escape arrow), or **clear-state-indicator** (as the result of an `)SIC` command).

If a line in a **defined-function** signals an error, **immediate-execution** is called to report the error and suspend the **defined-function**. **Immediate-execution** may return either a **branch** to indicate that evaluation of the **defined-function** should continue, or an **escape** or **clear-state-indicator** to indicate that the **defined-function** context should be removed from the **state-indicator**.

13.2 DEFINITIONS

Note: The following term defines the subset of arrays that can be used to create defined-functions.

Proper-Defined-Function: An array A is a proper-defined-function if

The type of A is character.

The rank of A is two.

A has at least one row.

The first row of A matches the character-diagram header-line.

No identifier in the first row of A is the distinguished-identifier $\square$ or $\square$ or is a system-function-name or niladic-system-function-name.

All rows after the first in A match the character-diagram body-line.

Note: The following definitions give properties of arrays that are proper-defined-functions.

Niladic: A , an array that is a proper-defined-function, is niladic if the form of the first row of A is niladic-function-header.

Monadic: A , an array that is a proper-defined-function, is monadic if the form of the first row of A is monadic-function-header.

Dyadic: A , an array that is a proper-defined-function, is dyadic if the form of the first row of A is dyadic-function-header.

Value-Returning: A , an array that is a proper-defined-function, is value-returning if, when header-line is threaded with the first row of A , the diagram result is threaded.

Note: The following terms categorise defined-functions according to their presence on or absence from the state-indicator.

Suspended: N , a simple-identifier, is suspended if it is the function-name of a context that immediately follows an immediate-execution context.

Pendent: N , a simple-identifier, is pendent if it is the function-name of a context that does not immediately follow an immediate-execution context.

Waiting: N , a simple-identifier, is waiting if a token whose class is defined-function-name and whose content is the content of N occurs in the stack of some context in the state-indicator of the active-workspace, and is not the first defined-function-name in that stack.

Note: For example, if F and G are defined function names, G is waiting during the evaluation of $(F\ 2)$ in the line $(F\ 2)\ G\ 3$.

Editable: N , a simple-identifier, is editable if the current-referent of N is nil, or if all the following are true:

The current-class of N is defined-function or niladic-defined-function.

The number-of-items in its referent-list is one.

N is neither pendent nor waiting.

N , if suspended, is suspended in only the current-context.

Note: This definition requires that a conforming-implementation be capable of editing the top function on the state indicator as long as it has not been localised and occurs nowhere else on the state indicator. This minimal definition is included to be certain that all conforming-implementations be capable of interactive program correction.

Globally-Erasable: Globally-Erasable is defined for N , a simple-identifier, as follows:

If no context of the state-indicator contains, in its current-statement, a defined-name with content that of N , then N is globally-erasable.

Otherwise, let L be the last (most global) context in whose local-name-list N appears.

If there is no such context then N is not globally-erasable.

Otherwise, let C be the last (most global) context in whose current-statement a defined-name with content that of N appears.

If the index of L is greater-than or equal to that of C then N is **globally-erasable**.

Note: Informally, N is **globally-erasable** if it was localised before it was made waiting or pendent.

Locally-Erasable: **Locally-Erasable** is defined for N , a **simple-identifier**, as follows:

If no **context** of the **state-indicator** contains, in its **current-statement**, a **defined-name** with **content** that of N , then N is **locally-erasable**.

Otherwise, let L be the first (most local) **context** in whose **local-name-list** N appears.

If there is no such **context** then N is not **locally-erasable**.

Otherwise, let C be the first (most local) **context** in whose **current-statement** a **defined-name** with **content** that of N appears.

If the index of C is greater than that of L , then N is **locally-erasable**.

Note: Informally, N is **locally-erasable** if it has been localised since the last time it was made waiting or pendent. This definition is used in $\square EX$ to specify that the expression $(\square EX'F') F 1$ be the same as $0 F 1$; it is used in $\square FX$ to specify that the expression $(\square FX 1 1p'F') F 1$ signal a domain-error.

Note: The following operations provide information about arrays that are **proper-defined-functions**.

Function-Name of A : For A a **proper-defined-function**, the **simple-identifier** that matches **function-name** when the **character-diagram header-line** is threaded with the first row of A .

Last-Line-Number of A : For A a **proper-defined-function**, negative-one plus the **number-of-rows** in A .

Header-Name-List of A : For A a **proper-defined-function**, a list of **identifiers** developed as follows:

Thread the **character-diagram header-line** with the first row of A .

Return a list of **tokens** consisting of **identifiers** that matched any of the **character-diagrams** **local-name**, **result-name**, **left-argument-name**, or **right-argument-name**.

Label-Name-List of A : For A a **proper-defined-function**, a list of **simple-identifiers** developed as follows:

Set Z to the **empty-list** of **tokens**.

For each **index** I in the **closed-interval-between one** and **last-line-number** of A ,

Thread the **character-diagram body-line** with row I plus one of A .

If **labelled-line** was matched, append the **simple-identifier token** that matched the **character-diagram label-name** to Z as a new last item.

Return Z .

Local-Name-List of A : For A a **proper-defined-function**, a list of **identifiers** consisting of the **label-name-list** followed by the **header-name-list**.

Identifier-Matching D in A : For A a **proper-defined-function** and D a **character-diagram**, a **simple-identifier token** developed as follows:

Thread the **character-diagram header-line** with the first row of A .

Return a **token** of class **simple-identifier** whose **content** is the list of **characters** that matched D .

Note: **Identifier-Matching** is used to refer to the arguments and result of an arbitrary defined function.

Function-Line I of A : For A a **proper-defined-function**, a list of **characters** developed as follows:

Thread the **character-diagram body-line** with row I plus one of A .

Return the list of **characters** that matched the **character-diagram line** in **body-line**.

Note: The following operations affect **symbols**.

Localise N :

Append a new first item, a token of class **nil**, to the **referent-list** of the **symbol-named-by N** .

Delocalise N :

If the **current-class** of N is **shared-variable**, **retract N** .

Remove the first item from the **referent-list** of the **symbol-named-by N** .

Current-Canonical-Representation: The **canonical-representation** of the **current-function** of the **current-context** in the **active-workspace**.

Current-Function-Line I : **Function-Line I** of the **current-canonical-representation**.

Current-Last-Line-Number: The **last-line-number** of the **current-canonical-representation**.

Current-Stop-Vector: The **stop-vector** of the **current-function** of the **current-context** in the **active-workspace**.

Current-Trace-Vector: The **trace-vector** of the **current-function** of the **current-context** in the **active-workspace**.

Current-Local-Names: The **local-name-list** of the **current-canonical-representation**.

Current-Right-Argument-Name: The **identifier-matching right-argument-name** in the **current-canonical-representation**.

Current-Left-Argument-Name: The **identifier-matching left-argument-name** in the **current-canonical-representation**.

Current-Result-Name: The **identifier-matching result-name** in the **current-canonical-representation**.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

13.3 DIAGRAMS

In these diagrams,

b stands for **permitted-blanks**.

r stands for **required-blanks**.

Header-Line

```
>>- b [ result ] b - form - b [ locals-list ] >>
```

Example:

$Z \leftarrow A \ F \ B; C; \square IO$

*Note: The header line indicates the name and syntactic class of the **defined-function**. It also gives a list of identifiers to be localised when the **defined-function** is called.*

This standard does not provide for end-of-line comments in header lines.

Result

```
>>----- result-name ----- b ----- assignment-arrow ----->>
```

Result-Name

```
>>----- simple-identifier ----->>
```

Form

```
>>-----> niladic-function-header ->
      |> monadic-function-header ->
      |> dyadic-function-header  ->----->>
```

Niladic-Function-Header

```
>>----- function ----->>
      -name
```

Monadic-Function-Header

```
>>----- function ----- r ----- right ----->>
      -name                -argument
                        -name
```

Dyadic-Function-Header

```
>>----- left ----- r --- function ----- r --- right ----->>
      -argument          -name              -argument
      -name              -name
```

Right-Argument-Name

```
>>———— simple-identifier ———>>
```

Left-Argument-Name

```
>>———— simple-identifier ———>>
```

Locals-List

```
>>_<_<_>>
    |_____|
    |      |
local — b — local — b
-marker   -name
```

Colon

Local-Marker

STANDARD

Function-Name

```
>>—— simple-identifier ——>>
```

Local-Name

```
>>—— identifier ——>>
```


Label-Name

>> — simple-identifier — >>

Body-Line

>> — b —> — labelled-line —> —>
 |
 -> — line —>

Labelled-Line

>> — label-name — b — colon — b — line —>>

Permitted-Blanks

>> —> —> —>>
 |
 -< — blank —<

Required-Blanks

>> —> — blank —> —>
 |
 <

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

13.4 OPERATIONS

13.4.1 Call-Defined-Function

$Z \leftarrow DFN$
 $Z \leftarrow DFN \ B$
 $Z \leftarrow A \ DFN \ B$

Informal Description: Z is the value of the variable given as the result name in the header line of the defined function whose name is DFN . If there is no result name, Z is **nil**.

Evaluation Sequence:

Generate a new **context** in which

mode is **defined-function**,
current-line is the empty list of characters,
current-function is the **current-referent** of DFN ,
current-line-number is **one**,
current-statement is the empty list of tokens, and
stack is the empty list of tokens.

Append the new **context** to the **state-indicator** of the **active-workspace** as a new first item.
Localise the **current-local-names**.

For form $Z \leftarrow A \ DFN \ B$

If **current-function** is not **dyadic**, signal **syntax-error**.
Set the **symbol-named-by current-left-argument-name** to A .
Set the **symbol-named-by current-right-argument-name** to B .

For form $Z \leftarrow DFN \ B$

If **current-function** is not **monadic**, signal **syntax-error**.
Set the **symbol-named-by current-right-argument-name** to B .

For all **indices** I in the **closed-interval-between one** and **current-last-line-number**,

If **row** I plus **one of canonical-representation** matches the diagram **labelled-line**, set the **current-referent** of the **simple-identifier** that matches **label-name** to a token of class **label** and **content** the **numeric-scalar** with value I .

Set Z to **defined-function-control**.
Delocalise the **current-local-names**.
Remove the **first-item** in the **state-indicator** of the **active-workspace**.
If Z is **escape**, signal **unwind**.
Otherwise, return Z .

Note: Tokens of class **escape** are converted to tokens of class **unwind** so that **immediate-execution** can distinguish an escape issued in **immediate-execution** mode from one issued in a **defined-function**.

The **context** attribute **current-function** includes the function itself plus its trace and stop vectors.

13.4.2 Defined-Function-Control

Form: Defined-Function-Control.

Informal Description: This procedure controls the execution of a user defined function.

Evaluation Sequence:

Set Z to a token whose class is nil.
 Let I stand for current-line-number.
 Set I to one.

Repeat:

If I is in the current-stop-vector, set attention-flag to one.
 If Z is an error or attention-flag is one,

Set N to immediate-execution with Z .
 If N is escape or clear-state-indicator, return N .
 If N is branch, set I to the content of N .

If I is not in the closed-interval-between one and current-last-line-number,

If the current-function is not value-returning, return nil.
 If the current-class of the current-result-name is variable or nil, return the current-referent of the current-result-name.
 Otherwise, signal value-error.

Set Z to evaluate-line of current-function-line I .
 If Z is escape, unwind, or clear-state-indicator, return Z .
 If Z is branch, set I to the content of Z .
 If Z is value or nil, set I to I plus one.

(End of repeated block)

Note: The attention-flag is reset by the user facility enter.

The class of the token returned by evaluate-line for $\rightarrow 10$ is nil, not branch.

The value-error signalled from defined-function-control is signalled in the line that invoked the defined function; the error is introduced to prevent conforming-programs from exiting a defined-function with the current-class of the result-name set to defined-function, niladic-defined-function, or shared-variable.

13.4.3 Function Fix

$Z \leftarrow \square FX B$

Informal Description: Z is, for B the **canonical-representation** of a defined function having **function-name** N , a **character** vector holding N . As a side effect, the defined function that B represents is established as the current referent of N . Rows of B that are all blanks give rise to blank lines in the established function.

Evaluation Sequence:

If the **rank** of B is not **two**, signal **rank-error**.

If B is an **empty array**, signal **length-error**.

If any **item** of the **ravel-list** of B is not a **character**, signal **domain-error**.

If B is not a **proper-defined-function**, signal **domain-error**.

If **identifiers** are duplicated in the **local-name-list** of B , signal **domain-error**.

Let N stand for the **simple-identifier** that matches **function-name** in the first **row** of B .

If the **current-class** of N is not **defined-function**, **niladic-defined-function**, or **nil**, signal **domain-error**.

If N is not **locally-erasable**, signal **domain-error**.

Set the **current-referent** of N to a **defined-function** for which

Canonical-Representation is B .

Stop-Vector is $\uparrow 0$.

Trace-Vector is $\uparrow 0$.

Return a **character vector** Z such that the **length** of Z is the **number-of-items** in the **content** of N and the **ravel-list** of Z is the **content** of N .

13.4.4 Character Representation

$$Z \leftarrow \square CR B$$

Informal Description: Z is a **character** matrix representation of the defined function named by B .

Evaluation Sequence:

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **length** of B is **zero**, signal **length-error**.

If B does not match the **character-diagram identifier-row**, signal **domain-error**.

If the **current-class** of B is not **defined-function** or **niladic-defined-function**, signal **domain-error**.

Otherwise return a **token** whose **class** is **constant** and whose **content** is the **canonical-representation** of the **current-content** of B .

Additional Requirements:

The formal model of APL used in this standard assumes that the internal representation of a defined function is precisely the **character** matrix satisfying the requirements for a **proper-defined-function** that was supplied as an argument to **function fix**. Representation of defined functions in this way in an implementation is neither required nor suggested.

The preservation of numeric literals and blanks as entered is desirable in the character representation, but it is not required by this standard. $\square CR \quad \square FX \quad \square CR \quad \square FX \quad X$ shall be the same as $\square CR \quad \square FX \quad X$, if X is a **proper-defined-function**.

If **numeric-literals** are stored as **numbers** in **defined-functions**, they are to be formatted with **print-precision** set to **full-print-precision**.

13.5 FUNCTION EDITING

Note: The function editing facilities described here are included to provide the APL programmer with a standard method for entering and correcting defined functions.

13.5.1 Evaluate-Function-Definition-Request

Evaluate-Function-Definition-Request Q .

Note: This subsection provides minimal facilities for the establishment and alteration of defined-functions.

An identifier can be edited by the function editing facilities only if it is **editable**. The header line of a **defined-function** can be altered only if the **function-name** is **locally-erasable**.

In **function-definition mode**, lines of the function being edited are temporarily associated with numbers in **decimal-rational** form by an unspecified mechanism. When a function is closed, the order of the lines given by the line number associated with each line is maintained.

The display of a function should be the same as the entries required to add the function to a **clear-workspace**.

This operation is called by **immediate-execution** when it recognises a **function-definition-line**.

Evaluation Sequence:

Thread **opening-request** with Q ; if Q does not match **opening-request**, signal **definition-error**.

Rethread **opening-request** with Q , taking the following actions:

When **creation-request** is matched,

Set G to the list of characters that matched **creation-request**.

Set N to the **simple-identifier** that matched **function-name** in G .

If the **current-class** of N is not **nil** and G does not match **identifier-row**, signal **definition-error**.

When **change-request** is matched,

Set N to the **simple-identifier** that matched the diagram **subject-function** in **change-request**.

Set G to the list of characters that matched **initial-request**.

If the **current-class** of N is **nil**, set M to a new **defined-function** for which

Canonical-Representation is 0 0p ' '

Stop-Vector is 1 0

Trace-Vector is 1 0.

Otherwise,

If N is not **editable**, signal **definition-error**.

Set M to the **current-referent** of N .

Generate a new **context** in which

mode is **function-definition**,

current-line is G ,

current-function is M ,

current-line-number is the **number-of-rows** in the **canonical-representation** of *M*,
current-statement is the empty **list of tokens**, and
stack is the empty **list of tokens**.

Append the **context** as a new first item to the **state-indicator**.

Repeat:

Set *Z* to **evaluate-editing-request**.

If *Z* is **command-complete**,

Remove the **first-item** in the **state-indicator** of the **active-workspace**.

Return *Z*.

If *Z* is an **error**, **display** *Z*.

Set **current-prompt** to **function-definition-prompt** of **current-line-number**.

Set **current-line** to **read-keyboard**.

(End of repeated block)

Additional Requirements:

The prompt in definition mode is a line number enclosed in brackets. Line numbers are always displayed in **decimal-rational** form. The line number of the header is always **zero**; it is not affected by **index-origin**.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

13.5.2 Evaluate-Editing-Request

Evaluate-Editing-Request

Informal Description: This subsection acts on a single line of input to the editor.

Evaluation Sequence:

Let C stand for the **current-canonical-representation**.
 Let N stand for the **simple-identifier** that matches **function-name** in **function-line**.
 Let S stand for the **current-stop-vector**.
 Let T stand for the **current-trace-vector**.
 Thread **general-request** with **current-line**; if **general-request** does not match **current-line**, signal **definition-error**.
 Otherwise, rethread **general-request** with **current-line**, taking the following actions:

When **positioning-request** is threaded,

Set L to the **numeric-input-conversion** of the list of **characters** that matched **line-number** in **positioning-request**.
 If L is a **negative-number**, signal **definition-error**.
 If L is **greater-than definition-line-limit**, signal **limit-error**.
 Set **current-line-number** to L .

When **deletion-request** is threaded,

Set L to the **numeric-input-conversion** of the list of **characters** that matched **line-number** in **deletion-request**.
 If L is not a **positive-number**, signal **definition-error**.
 If L is **greater-than definition-line-limit**, signal **limit-error**.
 Set **current-line-number** to L .
 Delete from C the row associated with L , if it exists.
 Delete from S and T the item associated with L , if it exists.

When **display-request** is threaded, **display function-display** of C .

When **function-line** is threaded, if **function-line** does not match **permitted-blanks**,

If **current-line-number** is zero,

If **function-line** does not match **header-line**, signal **definition-error**.
 If N is not **locally-erasable**, signal **definition-error**.

Set the row of C associated with **current-line-number** to the list of **characters** that matched **function-line**.
 If **current-line-number** is not an integer, or is **greater-than current-last-line-number**, set the items of S and T associated with **current-line-number** to zero.
 Set **current-line-number** to **next-definition-line** of **current-line-number**.

When **end-definition** is matched,

If C is not a **proper-defined-function**, signal **definition-error**.
 If **identifiers** are duplicated in the **local-name-list** of C , signal **definition-error**.
 Set the **current-referent** of N to a **defined-function** for which

Canonical-Representation is C .
Stop-Vector is S .
Trace-Vector is T .

Return **command-complete**.

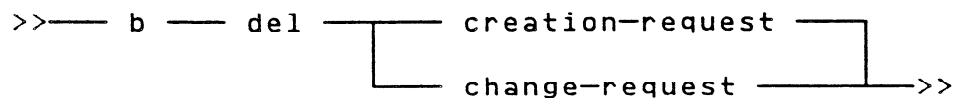
13.5.3 Diagrams

In these diagrams,

b stands for permitted-blanks.

r stands for **required-blanks**.

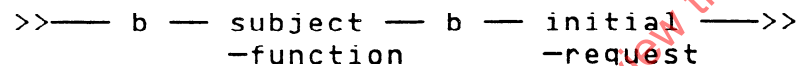
Opening-Request



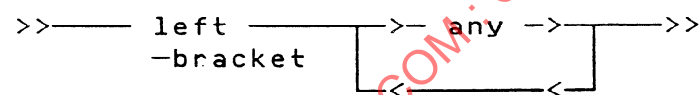
Creation-Request



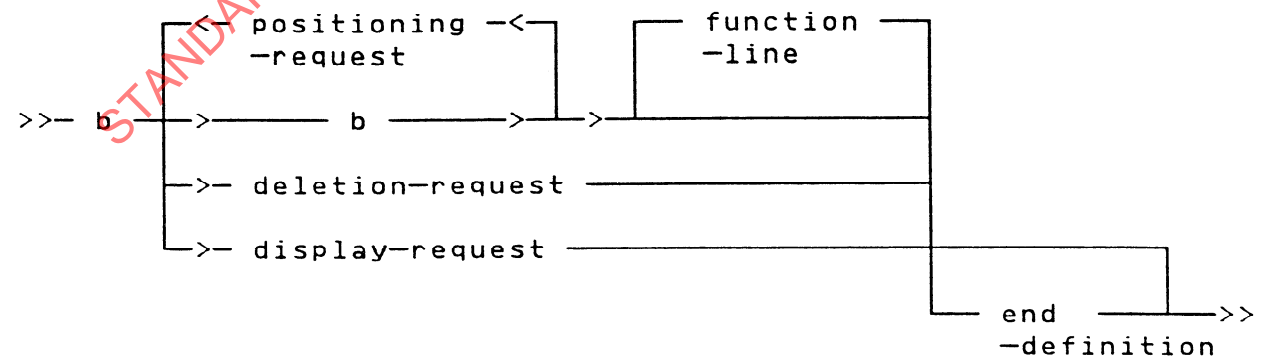
Change-Request



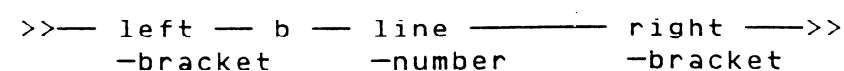
Initial-Request



General-Request



Positioning-Request



Deletion-Request

```
>>— left — b — delta — b — line ————— right —>>
      -bracket                -number      -bracket
```

Display-Request

```
>>— left — b — quad — b — right—>>
      -bracket                -bracket
```

Function-Line

>>— body-line —>>

End-Definition

>>— b — del — b —>>

Line-Number

```
>>— decimal-rational — b —>>
```

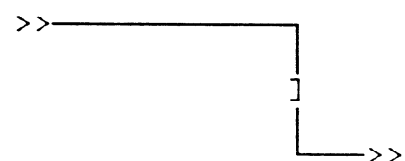
Subject-Function

>>— simple-identifier —>>

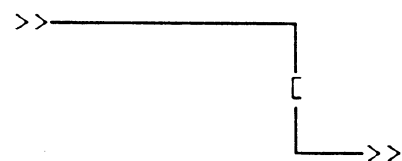
Delta



Right-Bracket



Left-Bracket



14 SHARED VARIABLES

14.1 INFORMAL INTRODUCTION

This section gives an informal introduction to the **Shared-Variable-Protocol**. The evaluation sequences which follow this introduction determine the required behaviour of **shared-variables** exactly.

The **Shared-Variable-Protocol** is an optional-facility.

The **Shared-Variable-Protocol** is described in this document as an interface between cooperating sessions, but it may also be used to provide an interface between a session and an auxiliary processor. An auxiliary processor is a program, written in any programming language, that typically provides access to facilities in the underlying operating system, such as file systems.

A **shared-variable** can be shared by precisely two partners, each either a session or an auxiliary processor. Conforming-programs that use the **Shared-Variable-Protocol** should document this fact, since the **Shared-Variable-Protocol** is not a defined-facility of this standard.

Operations

There are conceptually four operations involved in the **Shared-Variable-Protocol**: offer, retract, set, and reference.

Sharing

Offer is an operation provided to permit sessions to share **shared-variables**. A given **shared-variable** can be shared between at most two sessions. Once a **shared-variable** has been shared, assignments made to it in one session are visible in the other session.

In the evaluation sequences for the **Shared-Variable-Protocol**, sharing is accomplished by appending tokens of class **shared-variable** to the **shared-variable-list** which can be accessed by all **active-workspaces** in the system.

Any **Shared-Variable-Protocol** operation may cause a session to be delayed while another session changes a **shared-variable** or adds an item to the **shared-variable-list**.

Sharing begins when one session offers to share a symbol whose **current-class** is **nil** or **variable**. The system adds a new **shared-variable** to the **shared-variable-list** and returns the **shared-variable** to the offering session. The **shared-variable** then replaces the **current-referent** of the offered symbol.

There are two forms of offers. A **specific-offer** identifies a particular session as the partner; the **general-offer** does not.

Two offers match if both sessions use the same **shared-name** and:

At least one of the offers is a **specific-offer**.
Either **specific-offer** identifies the other session.

Consequently, two **general-offers** do not match.

Once both sessions have shared the **shared-variable**, the **shared-variable** is the **current-referent** of a symbol in both **active-workspaces**, and a change made to the **shared-variable** by one session will be visible to the other session.

Note that two sessions can share more than one **shared-variable**, that a given session can share **shared-variables** with more than one other session, and that each **shared-variable** can be shared by only two sessions.

Retracting

Two sessions, coupled by a **shared-variable**, decouple when either retracts the **shared-variable**. This leaves the **shared-variable** in the same state it would have been had the remaining session made a **specific-offer** to the session that retracted. If the remaining session retracts, the **shared-variable** becomes inaccessible to the sessions.

Degree of Coupling

All symbols in the **symbol-table** of an **active-workspace** have a **degree of coupling**. This is zero if the **current-referent** is not a **shared-variable**, one if the **current-referent** is a **shared-variable** that is offered to another session, and two if **current-referent** is a **shared-variable** shared with another session.

Naming

For any session, there are two identifiers associated with a given **shared-variable**. The first is the **name** of the symbol whose **current-referent** is the **shared-variable**; the second is the **shared-name** of the **shared-variable**. The **shared-name** is the name used in matching offers to share.

Only the **name** of the symbol is required for invoking any of the **Shared-Variable-Protocol** system functions other than **Shared-Variable Offer**.

Setting, Referencing, and Using

Each partner can set and reference the shared-variable.

A shared-variable is set when the operation **shared-variable-assignment** or **shared-variable-indexed-assignment** is called. A shared-variable is referenced when the operation **shared-variable-reference** is called. A shared-variable is used when it is either set or referenced. Both indexed assignment and indexed reference of shared-variables are possible.

Synchronisation

Synchronisation of sessions sharing a shared-variable is achieved by access control. With each shared-variable is associated an access-control-vector, which is designated here as the **ACV**. The **ACV** is a four element Boolean vector that is accessible to a session through the **Shared-Variable-Access-Control-Inquiry** operation.

The order of the elements in the **ACV** is relative to the viewing session. When session A, sharing a shared-variable with session B, views the **ACV**, its elements have the following meanings:

- If **ACV[1]** is one, once A has set the shared-variable B must use it before A can set it again.
- If **ACV[2]** is one, once B has set the shared-variable A must use it before B can set it again.
- If **ACV[3]** is one, once A has used the shared-variable B must set it before A can reference it again.
- If **ACV[4]** is one, once B has used the shared-variable A must set it before B can reference it again.

If a session attempts to use a shared-variable when not permitted to by the above rules, it is delayed until the partner has performed the required intermediate operation.

The rules can be summarised in a three-state state diagram. Let A be the session that made the initial offer. Then the states are as follows:

State 0: The shared-variable has been referenced by the partner of the session that last set it.

State 1: The shared-variable was last set by A and has not been referenced by B since then.

State 2: The shared-variable was last set by B and has not been referenced by A since then.

State 0 is the state entered when the shared-variable is initially offered. Note that only an explicit assignment is considered a set to the shared-variable. The act of transferring the current content of a symbol to the shared-value of a shared-variable when an offer is made is not a set. The state is not affected by retraction and re-sharing. Figure 2 summarises the state transitions permitted. In the figure, the action is always permitted if there is a + in the row corresponding to the input state. The column in which the + appears determines the output state. A digit in the row indicates that the corresponding element in the **ACV** as seen by A must be zero.

For example, if the state of a shared-variable is 0 and the **ACV** is 0 0 1 0, session A can assign a new value to the variable (SET by A), but must wait until the variable is in state 2 to refer to it (REF by A).

Note that all digits lie along the diagonal in these matrices, indicating that only actions that do not change the state can be blocked by the **ACV**. That is, the **ACV** can prevent only a given session from referencing or setting a shared-variable twice in a row.

next state:		0	1	2	0	1	2	0	1	2	0	1	2
		.	.	.	.	.	.	.	.	.	.	.	.
		.	.	.	.	.	.	.	.	.	.	.	.
cur-	0	...	.	+	.	.	.	3	.	.	4	.	.
rent	1	...	.	1	.	.	.	.	3	.	+	.	.
state	2	...	.	+	.	.	2	+	.	.	.	.	4
action		set			set			ref			ref		
		by			by			by			by		
attempted:		a			b			a			b		

Figure 2 — Shared Variable Access Rules

Other Rules

Retraction of all the **shared-variables** in a **session** takes place when a new **workspace** is loaded, the **active-workspace** is cleared, or the **session** ends. A **shared-variable** is automatically **retracted** when the **shared-variable** is erased, expunged, or replaced by a **)COPY** command, or, if it is local, when the function to which it is local terminates execution. The state of sharing is not saved in a library workspace.

The shared variable mechanism must appear to the user to be consistent with one in which there exists a single copy of each **shared-variable**. Specifically, this means:

When a **shared-variable** is initially offered, its value, if any, becomes the **shared-value** of the **shared-variable**.

When an offered **shared-variable** is matched, it retains its value if it has one. If it does not have a value, it takes on the value (if any) in the matcher's workspace.

When a **shared-variable** is retracted, its value in the retractor's workspace is the value last set by either partner before the retraction.

Note that, since there is conceptually a single copy across a period of sharing, no use is implied in the above rules for making the value of a **shared-variable** available.

It must appear to the APL user that **sessions** communicating via **shared-variables** are independently scheduled. Any synchronisation between the **sessions** must appear to be by means of the delays caused by accessing **shared-variables**. The procedures used to describe the **Shared-Variable-Protocol** are written as though synchronisation conflicts do not occur when two **sessions** are accessing the **shared-variable-list** or the same **shared-variable**.

A **limit-error** may occur during the retraction of an active **shared-variable** if there is not enough room in a **workspace** for the **shared-value** of the **shared-variable**. When the expected system action is to ignore the variable (as in the case of **)CLEAR** or delocalisation), a **limit-error** is not signalled.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

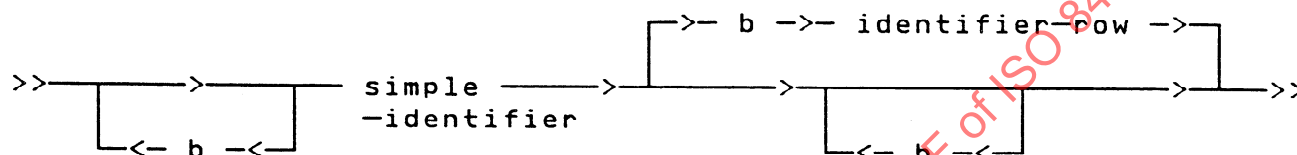
14.2 DEFINITIONS

Identifier-Pair-Array: A character array of rank two in which each row matches the character-diagram identifier-pair-row.

14.3 DIAGRAMS

In the following, b stands for **blank**.

Identifier-Pair-Row



14.4 OPERATIONS

14.4.1 Primary-Name

Primary-Name in A : For A , a character vector that matches identifier-pair-row, the first simple-identifier matched in identifier-pair-row.

14.4.2 Surrogate-Name

Surrogate-Name in A : For A , a character vector that matches identifier-pair-row, the last simple-identifier matched in identifier-pair-row.

14.4.3 Degree-of-Coupling

Degree-of-Coupling of N : For symbol N , a number defined as follows:

If the **current-class** of N is not **shared-variable**, **zero**.

Otherwise, using the **current-referent** of N , the sum of **session-A-active** and **session-B-active**.

14.4.4 Access-Control-Vector

Access-Control-Vector of N : An operation that, for N a **shared-variable**, is defined as follows:

Using N ,

Let $ACVA$ stand for **session-A-ACV** and $ACVB$ stand for **session-B-ACV**.

If **this-session** is **session-A**, return $ACVA \vee ACVB$.

If **this-session** is **session-B**, return $(ACVA \vee ACVB)[2 \ 1 \ 4 \ 3]$.

14.4.5 Offer

Offer S to P with value V : An operation defined as follows:

Let X stand for the **item** in the **shared-variable-list** with the smallest **index** such that

shared-name is S .
session-A is **this-session** or, if P is not **general-offer**, **general-offer**.
session-A-active is **zero**.
session-B is P .
session-B-active is **one**.

or

shared-name is S .
session-A is P .
session-A-active is **one**.
session-B is **this-session** or, if P is not **general-offer**, **general-offer**.
session-B-active is **zero**.

If such a **shared-variable** exists,

If **shared-value** is **nil**, set **shared-value** to V .

If **session-A-active** is **zero**,

Set **session-A-active** to **one**.
Set **session-A** to **this-session**.

If **session-B-active** is **zero**,

Set **session-B-active** to **one**.
Set **session-B** to **this-session**.

Return X .

Otherwise, create a new **shared-variable**, Y , as follows:

shared-name is S .
session-A is **this-session**.
session-A-active is **one**.
session-A-ACV is 0 0 0 0.
session-B is P .
session-B-active is **zero**.
session-B-ACV is 0 0 0 0.
shared-value is V .
state is **zero**.

Append Y to the **shared-variable-list** as a new last **item**.

Return Y .

14.4.6 Retract

Retract N : An operation that, for N a **shared-variable**, is defined as follows:

Using N ,

If **session-A** is **this-session**, set **session-A-active** to zero.
If **session-B** is **this-session**, set **session-B-active** to zero.

14.4.7 Shared-Variable-Reset

Shared-Variable-Reset

For each item in the **shared-variable-list**,

If **session-A** is **this-session**, set **session-A-active** to zero.
If **session-B** is **this-session**, set **session-B-active** to zero.

For each **symbol N** of the **symbol-table** of the **active-workspace**,

For each **token T** in the **referent-list** of N ,

If T is a **shared-variable**, set T to a **token** whose **class** is **nil**.

*Note: In the model, a **shared-variable** is never reused. It is effectively discarded once **session-A-active** and **session-B-active** are both zero.*

14.5 SHARED VARIABLE FORMS

14.5.1 Shared Variable Reference

$$Z \leftarrow SHV$$

Informal Description: Return the value of a shared variable.

Evaluation Sequence:

Using the **current-content** of *SHV*,

Let *ACV* stand for **access-control-vector**.
Wait until at least one of the following is true:

ACV[3] is **zero**.
State is **two** and **session-A** is **this-session**.
State is **one** and **session-B** is **this-session**.

If either of the following is true, set **State** to **zero**:

State is **two** and **session-A** is **this-session**.
State is **one** and **session-B** is **this-session**.

Return **Shared-Value**.

Note: This operation is all that is required for **shared-variable-indexed-reference**, since the **phrase-evaluator** calls **indexed reference** with a value, not a name.

14.5.2 Shared Variable Assignment

$$Z \leftarrow SHV \leftarrow B$$

Informal Description: *Z* is *B*. As a side effect, the value *B* is assigned to the shared variable *SHV*.

Evaluation Sequence:

Using the **current-content** of *SHV*,

Let *ACV* stand for **access-control-vector**.
Wait until at least one of the following is true:

ACV[1] is **zero**.
State is not **one** and **session-A** is **this-session**.
State is not **two** and **session-B** is **this-session**.

Set **shared-value** to *B*.

If **session-A** is **this-session**, set **state** to **one**.

If **session-B** is **this-session**, set **state** to **two**.

Return a **token** whose **class** is **committed-value** and whose **content** is *B*.

14.5.3 Shared Variable Indexed Assignment

$$Z \leftarrow SHV[I] \leftarrow B$$

Informal Description: Z is B . As a side effect, elements of the array SHV selected by the position and values of the arrays in the **index-list** I are replaced by elements of the array B .

Evaluation Sequence:

Using the **current-content** of SHV ,

Let ACV stand for **access-control-vector**.
Wait until at least one of the following is true:

$ACV[1]$ is **zero**.
session-A is **this-session** and **state** is not **one**.
session-B is **this-session** and **state** is not **two**.

If the **current-class** of **shared-value** is **nil**, signal **value-error**.

Set C to **shared-value**.

Search the **form-table** for $Z \leftarrow V[I] \leftarrow B$.

Call the corresponding evaluation sequence, passing C as the value of V .

Set Z to the **token** it returns.

If Z is an **exception**, return Z .

Otherwise,

Set **shared-value** to C .

If **session-A** is **this-session**, set **state** to **one**.

If **session-B** is **this-session**, set **state** to **two**.

Return Z .

Note: Indexed assignment of a shared variable constitutes a **set**, not a **reference**, of the **shared-variable**.

14.6 SHARED VARIABLE SYSTEM FUNCTIONS

14.6.1 Shared Variable Access Control Inquiry

$$Z \leftarrow \square SVC \ B$$

Informal Description: Each row of Z is the **access-control-vector** of the shared variable named by the corresponding row of B .

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Set $B1$ to $(\neg 2 \uparrow 1 \ 1, \rho B) \rho B$.

If any **item** of the **ravel-list** of $B1$ is not a **character**, signal **domain-error**.

Set Z to $((1 \uparrow \rho B1), 4) \rho 0$.

For every I in $1 \uparrow \rho B1$, evaluated with **index-origin** set to **one**,

Let $B2$ stand for **row** I of $B1$.

If $B2$ matches **identifier-row**,

Set N to the **simple-identifier** in $B2$.

If the **current-class** of N is **shared-variable**,

Set **row** I of Z to **access-control-vector** of the **current-referent** of N .

Otherwise, signal **domain-error**.

Return $((\neg 1 \uparrow \rho B), 4) \rho Z$.

14.6.2 Shared Variable Query

$$Z \leftarrow \square SVQ \ B$$

Informal Description: Z is one of the following:

An array representing sessions offering to share variables with this session.

An array representing the names of shared variables offered to this session by another session.

Evaluation Sequence:

If **session-identification-type** is **character**, and B is the **general-offer**,

Set Z to 0 0 ρ^1 1.

Using each item of the **shared-variable-list**,

If

session-A is **this-session** and
session-A-active is zero and
session-B-active is one,

Set S to **session-B**

Set Z to $((\rho Z) \uparrow 0, \rho S), [1] ((\neg 1 \uparrow \rho Z) \uparrow \rho S) \uparrow S$, evaluated with
index-origin set to **one**.

If

session-A-active is one and
session-B is **this-session** and
session-B-active is zero,

Set S to **session-B**

Set Z to $((\rho Z) \uparrow 0, \rho S), [1] ((\neg 1 \uparrow \rho Z) \uparrow \rho S) \uparrow S$, evaluated with
index-origin set to **one**.

Return Z .

If **session-identification-type** is **numeric**, and B is **zero**, signal **domain-error**.

If **session-identification-type** is **numeric**, and B is 1 0,

Set Z to 1 0.

Using each item of the **shared-variable-list**,

If all of the following are true,

session-A is **this-session**
session-A-active is zero
session-B-active is one

append **session-B** as a new last element to the **vector** Z .

If all of the following are true,

session-A-active is one
session-B is **this-session**
session-B-active is zero

append **session-A** as a new last element to the **vector** Z .

Return $((Z_1 Z) = 1 \rho Z) / Z$.

Otherwise,

If B is not a **session-identification**, signal **domain-error**.
 If B is **this-session**, signal **domain-error**.
 Set Z to the **empty character array** of rank **two**.
 Using each **item** of the **shared-variable-list**,

If all of the following are true,

session-A is this-session
session-A-active is zero
session-B is B
session-B-active is one

append **shared-name** as a new last row to the **identifier-array Z** .

If all of the following are true,

session-A is B
session-A-active is one
session-B is this-session
session-B-active is zero

append **shared-name** as a new last row to the **identifier-array Z** .

Return Z .

14.6.3 Shared Variable Degree of Coupling

$Z \leftarrow \square SVO\ B$

Informal Description: Z is a numeric vector in which each element is the degree of coupling of the symbol named by the corresponding row of B .

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Set $B1$ to $(\neg 2 \uparrow 1\ 1, \rho B) \rho B$.

If any **item** of the **ravel-list** of $B1$ is not a **character**, signal **domain-error**.

Set Z to $(1 \uparrow \rho B1) \rho 0$.

For every I in $1 \uparrow \rho B1$,

Let $B2$ stand for **row I of $B1$** .

If $B2$ matches **identifier-row**,

Set N to the **primary-name** in $B2$.

Set **item I of the ravel-list of Z** to **degree-of-coupling of the symbol-named-by N** .

Otherwise, signal **domain-error**.

Return Z .

Note: *Degree-of-coupling is a property of symbols, not shared-variables.*

14.6.4 Shared Variable Offer

$Z \leftarrow A \square SVO B$

Informal Description: Z is a numeric vector representing the **degree-of-coupling** of the identifiers in the rows of the **identifier-pair-array** B after the operation. As a side effect, dyadic $\square SVO$ shares variables with another **session**. B is an **identifier-pair-array** representing the names of variables to be shared. A is a **session-identification**, representing the **session** with which the names in B are to be shared.

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

If any **item** of the **ravel-list** of B is not a **character**, signal **domain-error**.

If A is not a valid **session-identification**, signal **domain-error**.

If **this-session** is A , signal **domain-error**.

Set $B1$ to $(\neg 2 \uparrow 1 \ 1, \rho B) \rho B$.

For every I in $1 \uparrow \rho B1$,

Let $B2$ stand for **row** I of $B1$.

If $B2$ does not match **identifier-pair-row**, signal **domain-error**.

Otherwise,

Set N to the **primary-name** in $B2$.

Set S to the **surrogate-name** in $B2$.

If the **current-class** of N is neither **shared-variable**, **variable** nor **nil**, signal **domain-error**.

If the **current-class** of N is **variable** or **nil**,

Set V to the **current-referent** of N .

Set Y to **offer** S to A with **value** V .

Set the **current-referent** of N to Y .

Return $\square SVO B$.

Note: This operation does not allow for multiple **session-identifications**.

14.6.5 Shared Variable Retraction

$$Z \leftarrow \square SVR\ B$$

Informal Description: Elements of Z hold the degree of coupling that the identifier represented in the corresponding row of the **identifier-array** B had, before this retraction. As a side effect, $\square SVR$ retracts any shared-variables named in B .

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Set $B1$ to $(\neg 2 \uparrow 1\ 1, \rho B) \rho B$.

Set Z to $\square SVO\ B1$.

If Z is an **error**, return Z .

For every I in $1 \uparrow 1 \uparrow \rho B1$,

Let $B2$ stand for **row** I of $B1$.

If $B2$ matches **identifier-row**,

Set N to the **simple-identifier** in $B2$.

If the **current-class** of N is **shared-variable**,

Set SV to the **current-referent** of N .

Set the **current-referent** of N to **shared-value** of SV .

Retract SV .

Otherwise, signal **domain-error**.

Return Z .

14.6.6 Shared Variable Access Control Set

$$Z \leftarrow A \sqcup SVC \ B$$

Informal Description: Z is the **access-control-vector** of each of the shared variables represented by the **identifier-array** B when $\sqcup SVC$ completes. As a side effect, $\sqcup SVC$ sets to A the contribution this session makes to the combined access control vector of the shared variables.

Evaluation Sequence:

If the **rank** of B is **greater-than two**, signal **rank-error**.

Set $B1$ to $(\neg 2 \uparrow 1 \ 1, \rho B) \rho B$.

If the **rank** of A is **greater-than two**, signal **rank-error**.

If the first item of $(\neg 1 \uparrow 4, \rho A)$ is not **four**, signal **length-error**.

If A is a **scalar** or a **vector**, set $A1$ to $((1 \uparrow \rho B1), 4) \rho A$.

Otherwise, set $A1$ to A .

If $1 \uparrow \rho A1$ is not the same as $1 \uparrow \rho B1$, signal **length-error**.

If any item of the **ravel-list** of $B1$ is not a **character**, signal **domain-error**.

If any item of the **ravel-list** of $A1$ is not a **near-Boolean**, signal **domain-error**.

Set $A2$ to the **Boolean-array-nearest-to** $A1$.

For every I in $1 \uparrow \rho B1$,

Let $B2$ stand for **row** I of $B1$.

If $B2$ matches **identifier-row**,

Set N to the **simple-identifier** in $B2$.

If the **current-class** of N is **shared-variable**,

Let ACV stand for **row** I of $A2$.

Using the **current-content** of N ,

If **session-A** is **this-session**, set **session-A-ACV** to ACV .

If **session-B** is **this-session**, set **session-B-ACV** to $ACV[2 \ 1 \ 4 \ 3]$.

Otherwise, signal **domain-error**.

Return $\sqcup SVC \ B$.

15 FORMATTING AND NUMERIC CONVERSION

15.1 INTRODUCTION

The mapping of arrays of numbers into arrays of characters is referred to as formatting.

*Note: The form, field width, and number of digits in a formatted number can be specified explicitly through the use of **dyadic format**, or left to be chosen by the system. **Conforming-programs** that depend upon a particular selection of formatting parameters should employ **dyadic format**.*

15.2 NUMERIC CONVERSION

*Note: Conversion involves the transformation of lists of **characters** to and from **numbers**. This conversion is performed by the **implementation-algorithms numeric-input-conversion** and **numeric-output-conversion**.*

The accuracy and rounding properties of **numeric-input-conversion** and **numeric-output-conversion** are interdependent. The most significant property of these **implementation-algorithms** is the following:

For any **numeric scalar** X , when **print-precision** is set to **full-print-precision**, X shall be the same as $\text{to } \Phi X$.

Informal Description: Numeric input conversion converts numeric values represented in decimal notation as **lists of characters** into equivalent **numbers** – numeric quantities whose format is **implementation-defined**.

Let L and G be respectively the most negative and the most positive abstract numeric values represented by all **numeric-scalar-literals** producible by **numeric-output-conversion** for all **numbers** and all valid values of **print-precision**.

Let $\mathcal{T}$ be the set of all lists of characters that match the diagram **numeric-scalar-literal** and have abstract numeric values lying in the closed interval between L and G .

Otherwise,

Let c stand for the abstract numeric value of C .

If **c** is the same as some **number** *N*, return *N*.

If `c` is greater than `positive-number-limit`, return `positive-number-limit`.

If `c` is less-than `negative-number-limit`, return `negative-number-limit`.

Otherwise, c lies between two numbers. Return one that permits numeric-input-conversion to satisfy the following property:

For any two **numeric-scalar-literals** $C1$ and $C2$, if the abstract numeric value of $C1$ is less than the abstract numeric value of $C2$, then **numeric-input-conversion** of $C1$ is not **greater-than numeric-input-conversion** of $C2$.

0

[illegible]

0

1-0.999

1

[illegible]

0

0E999

1

$$1E^{-9999} = 0$$

0

2.5E4-25000

Every number can be obtained by converting some numeric-scalar-literal.

There is no number N for which the expression $\Phi \Phi N$ signals a limit-error.

15.2.2 Numeric-Output-Conversion

E Numeric-Output-Conversion of *N*.

E Numeric-Output-Conversion of *N* to *P* places.

Informal Description: Numeric output conversion converts numeric values represented as **numbers** – numeric quantities whose format is **implementation-defined** – into the same numeric quantities represented in decimal notation as **lists of characters**.

For *E* a **character-diagram**, and *N* and *P* **numbers**, numeric output conversion returns a **numeric-scalar-literal** that matches *E* and represents the abstract numeric value of *N* either to *P* decimal places, if *P* is specified, or else to **print-precision** places.

Evaluation Sequence:

If *E* is **minimal-decimal-exponential**,

Return a **list of characters** whose abstract numeric value is a good approximation of *N*.

Note: The choice of approximation is **implementation-defined**, but can be characterised as follows:

Let *DS* be the set whose members *D* are lists of characters that match **minimal-decimal-exponential** and have at most **print-precision** digits to the left of the **exponent-marker**.

Let *DT* be the subset of *DS* for which $|N - \text{value}(D)|$ is minimal.

Let *DM* be the subset of *DT* whose members have the fewest digits.

Choose the member of *DM* whose abstract numeric value has the largest magnitude.

If *E* is **fixed-decimal**,

Let *DF* be the set whose members are lists of characters that match **fixed-decimal** and have *P* digits to the right of the units digit.

Return a member *D* of *DF* for which $|N - \text{value}(D)|$ is minimal.

If *E* is **decimal-exponential**,

Let *DE* be the set whose members are lists of characters that match **decimal-exponential** and have *P* digits to the left of the **exponent-marker**.

Select a member *D* of *DE* for which $|N - \text{value}(D)|$ is minimal.

Let *W* stand for **exponent-field-width**.

Return $(W + D1 'E') \uparrow D$.

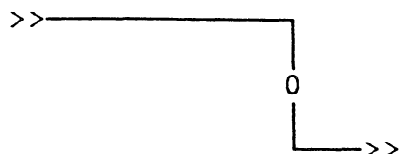
Note: When *P* is not specified, *E* is always **minimal-decimal-exponential**.

Fixed-decimal and **decimal-exponential** cases are used only by **dyadic format**. They are permitted, but not required, to return results of arbitrarily high precision. They should, however, return accurate results if no more than **full-print-precision** significant digits are requested.

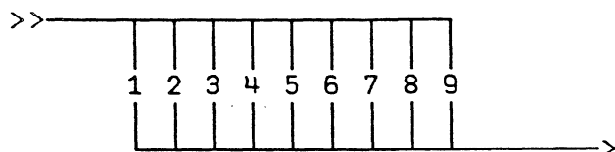
15.3 DIAGRAMS

Note: The following **character-diagrams** characterise the outputs of both **monadic** and **dyadic format** as subsets of the set of lists of characters that match **numeric-scalar-literal**.

Zero-Digit



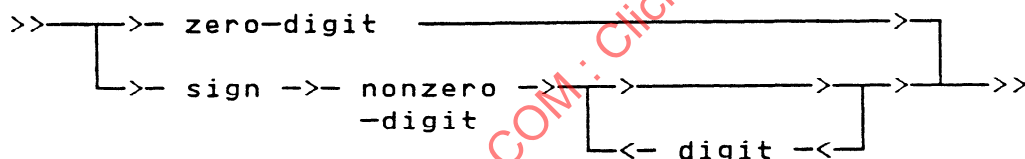
Nonzero-Digit



Sign



Decimal-Integer

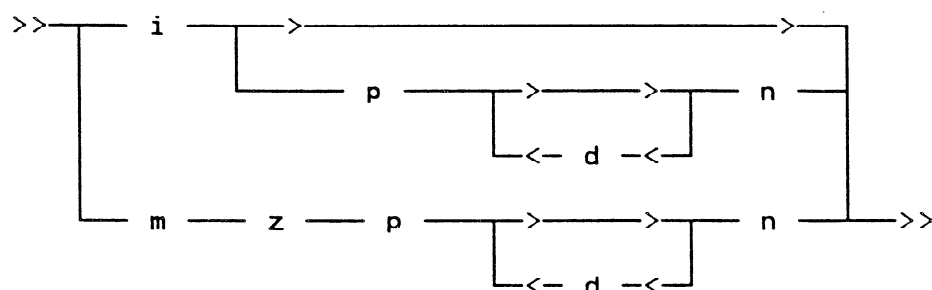


Examples:

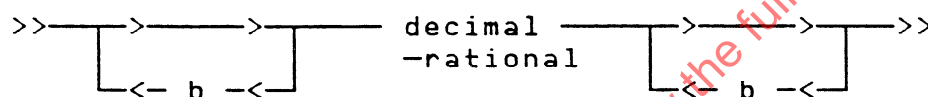
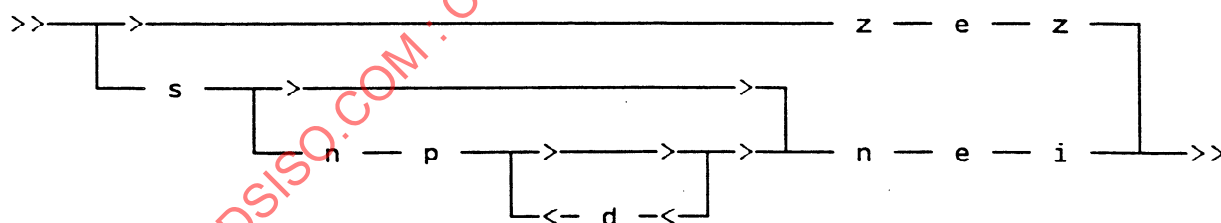
`-44 0 622 1066 1215 1415 1685 1750 1776 1812 1867 1944`

In the following diagrams,

- b stands for **blank**.
- d stands for **digit**.
- e stands for **exponent-marker**.
- i stands for **decimal-integer**.
- m stands for **overbar**.
- n stands for **nonzero-digit**.
- p stands for **dot**.
- s stands for **sign**.
- z stands for **zero-digit**.

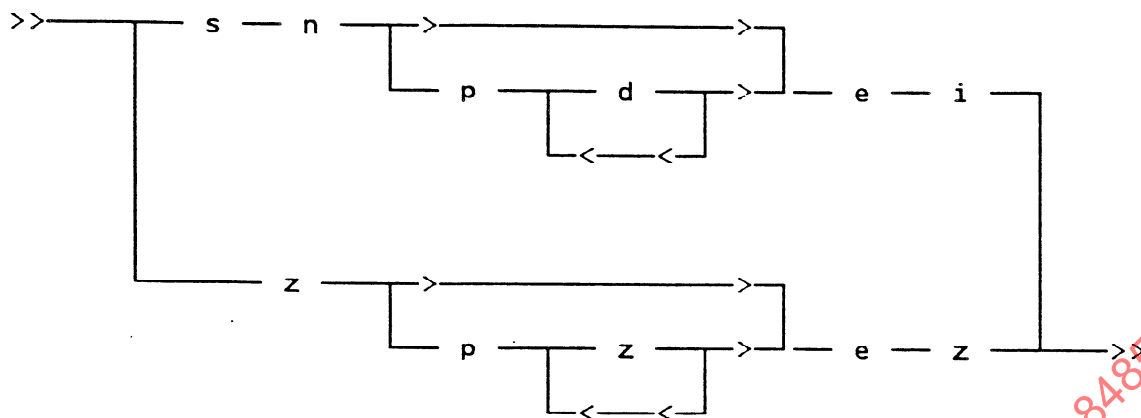
Decimal-Rational**Examples:**

98.6 212 $\bar{2}.001$ $\bar{0}.000000000001$

Decimal-Rational-Row**Minimal-Decimal-Exponential****Examples:**

$\bar{1}.234567E^{-890}$ $1E1$ $\bar{1}.1111E1$ $1E^{-11111}$ $0E0$

Decimal-Exponential



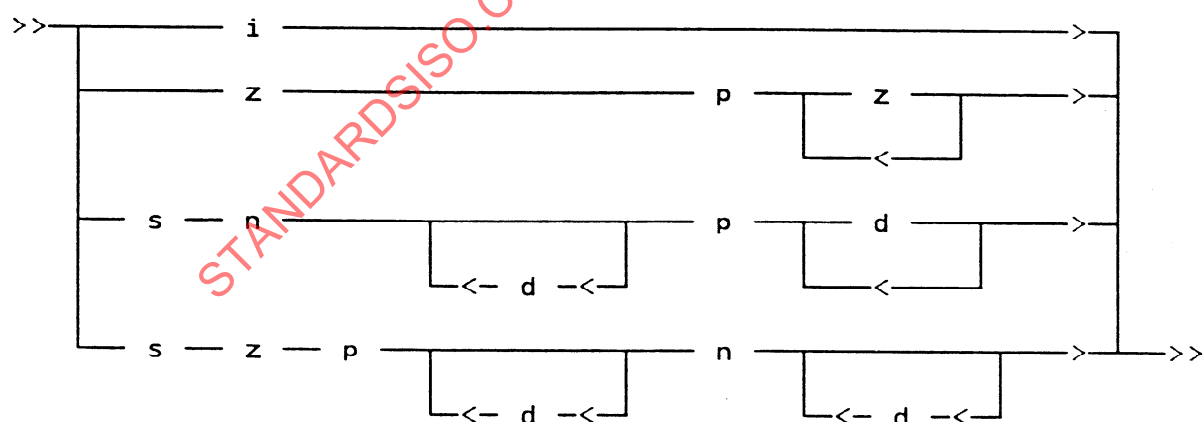
Examples:

$-1.23456700000E^{-890}$ $0.000000000E0$ $-1.0000E^{-10}$

Decimal-Exponential-Row



Fixed-Decimal



Examples:

0^{-1} 12 0.000000000 12.30 0.1235 -0.1235 0.1 -0.00010000

15.4 OPERATIONS

15.4.1 Monadic Format

$$Z \leftarrow \text{f} B$$

Informal Description: Z is a **character** array. If B is a **character** array, then Z is B . If B is a numeric array, then Z is a **character** array arranged so that each row of Z is a decimal representation of the corresponding row of B . The output form chosen for a column of B is determined by **print-precision** and the values of elements in that column. Uses **print-precision**.

Evaluation Sequence:

If **print-precision** is nil, signal **implicit-error**.

If the **type** of B is **character**, return B .

If B is **empty**, return an **array** $Z0$ such that the **type** of $Z0$ is **character**, the **shape-list** of $Z0$ is the **shape-list** of B , and the **ravel-list** of $Z0$ is the **empty-list**.

If the **rank** of B is **less-than two**,
return $\text{f}(\text{f}^{-2} \uparrow 1 \text{ } 1, \rho B) \rho B$.

If the last **item** of the **shape-list** of B is not **one**,
return $(\text{f}(\text{f}^{-1} \uparrow \rho B), 1) \uparrow B, ' ', \text{f}((- \rho \rho B) \uparrow 1) \uparrow B$.

Otherwise, choose U , E , and W , and return an **array** Z for which the **type** of Z is **character**, the **shape-list** of Z is $(\text{f}^{-1} \uparrow \rho B), W$, and one of the following statements is true:

Every **row** of Z is the **decimal-exponential-row** equivalent of the **minimal-decimal-exponential numeric-output-conversion** of the corresponding **row** of B , and has its **exponent-marker** at position E and its first digit at position U .

Every **row** of Z is the **decimal-rational-row** equivalent of the **minimal-decimal-exponential numeric-output-conversion** of the corresponding **row** of B , and has its units digit at position U .

Note: The quantities W , U and E and the choice of formatting form are not standardised here, since this is currently impractical. However, the following algorithm is recommended for all future implementations of APL:

If any of the following conditions holds for any element X of B

X is greater-than positive-counting-number-limit.

X is less-than negative-counting-number-limit.

The decimal-rational equivalent of the minimal-decimal-exponential numeric-output-conversion of X would have more than print-precision significant digits to the left of the decimal point.

The decimal-rational equivalent of the minimal-decimal-exponential numeric-output-conversion of X would have more than five zero digits to the right of the decimal point and to the left of the first nonzero digit.

then select decimal-exponential form.

Otherwise, select decimal-rational form.

If decimal-exponential form is selected,

Let S be one if any element of B is a negative-number.

Let M be the maximum number of characters to the left of the exponent-marker in the minimal-decimal-exponential numeric-output-conversion of any element of B .

Let N be the maximum number of characters to the right of the exponent-marker in the minimal-decimal-exponential numeric-output-conversion of any element of B .

Set U to $S+1$.

Set E to $M+1$.

Set W to $M+1+N$.

If decimal-rational form is selected,

Let M be the maximum number of characters to the left of the units position in the decimal-rational equivalent of the minimal-decimal-exponential numeric-output-conversion of any number in B .

Let N be the maximum number of characters to the right of the units position in the decimal-rational equivalent of the minimal-decimal-exponential numeric-output-conversion of any number in B .

Set U to $M+1$.

Set W to $M+1+N$.

Examples:

 $\square PP \leftarrow 10$

		' ', (5 ⁻⁶ 7 8 9 10 ⁰ ÷ 15), ' '	
5	2.5	1.666666667	1.25 1
-6	-3	-2	-1.5 -1.2
7	3.5	2.333333333	1.75 1.4
8	4	2.666666667	2 1.6
9	4.5	3	2.25 1.8
10	5	3.333333333	2.5 2

$$D \leftarrow 0.7 \quad 0.8 \quad 0.9 \quad 7 \quad 8 \quad 9 \div 10$$
 $\Phi \quad D$

0.7 0.8 0.9 0.7 0.8 0.9

$$D - \Phi \Phi^* D$$

```

0 0 0 0 -1.387778781E-17 0

```

□PP←999

 ΦD

0.7 0.8 0.9 0.7 0.799999999999999999 0.9

$$D - \Phi \Phi^* D$$

0 0 0 0 0 0

 $\square PP \leftarrow 4$
$$M \leftarrow 0.78901 \times (10 \times 16)^{0.5} \times 1 \quad 1E-10 \quad 1E-6 \quad 0.01 \quad 0.1$$
 $\Phi \quad M$

$7.89E0$	$7.89E^{-10}$	0.00000789	0.0789	$7.89E^{-1}$
$7.89E1$	$7.89E^{-9}$	0.0000789	0.789	$7.89E0$
$7.89E2$	$7.89E^{-8}$	0.000789	7.89	$7.89E1$
$7.89E3$	$7.89E^{-7}$	0.00789	78.9	$7.89E2$
$7.89E4$	$7.89E^{-6}$	0.0789	789	$7.89E3$
$7.89E5$	$7.89E^{-5}$	0.789	7890	$7.89E4$

Φ 50M

7.89 7.89E-10 0.00000789 0.0789 0.789

15 μM

7.89 7.89E-10 0.000000789 0.0789 0.789

Note: When *print-precision* has a value in its *internal-value-set* greater than or equal to *full-print-precision*, the result produced for N is unique and is unaffected by increases in the value of *print-precision*. At such settings of *print-precision*, N is the same as $\oplus \oplus N$ for any numeric scalar N .

15.4.2 Dyadic Format

$$Z \leftarrow A \oplus B$$

Informal Description: Z is a **character** array representing the numbers in B formatted according to the specifications in A . A consists of pairs of integers. One pair is associated with each column of B . The first integer in the pair specifies the field width – the number of columns in the character result – for the formatted numeric values.

If A has only two elements then all elements of B are formatted according to A .

A non-negative second integer indicates that the field is to have **fixed-decimal** form and gives the number of digits to the right of the decimal point, with zero indicating no decimal places and no decimal point. A negative second integer indicates that the field is to have decimal exponential form; the absolute value specifies the number of digits in the mantissa.

Evaluation Sequence:

If the **rank** of A is **greater-than one**, signal **rank-error**.

If ρ, A is not $2 \times^{-1} + 1, \rho B$

If ρA is **two**, return $((2 \times^{-1} + \rho B) \rho A) \oplus B$.

Otherwise, signal **length-error**.

If any **item** of the **ravel-list** of A is not a **near-integer**, signal **domain-error**.

If any **item** of the **ravel-list** of B is not a **number**, signal **domain-error**.

Set $A1$ to the **integer-array-nearest-to** A .

If B is **empty**,

Set W to $+ / ((\rho A1) \rho 1 \ 0) / A1$.

Return $((^{-1} + \rho B), W) \rho ' '$.

If $\rho A1$ is not **two**, return $((2 + A1) \oplus ((^{-1} + \rho B), 1) + B), (2 + A1) \oplus ((-\rho \rho B) + 1) + B$.

If B is not a **scalar**,

If $A1[1]$ is not a **positive-integer**, signal **domain-error**.

Return a **character array** Z such that the **shape-list** of Z is $(^{-1} + \rho B), A1[1]$ and the **ravel-list** of Z has the property that each **vector-item along-axis** $\rho \rho Z$ of Z is $A1 \oplus B0$, where $B0$ is the corresponding (scalar) **item** of $(^{-1} + \rho B) \rho B$.

If B is a **scalar**,

If $A1[2]$ is not **less-than zero**, set R to the **fixed-decimal numeric-output-conversion** of B to $A1[2]$ places.

Otherwise, set R to the **decimal-exponential numeric-output-conversion** of B to $|A1[2]|$ places.

If $A1[1]$ is **less-than** ρR , signal **domain-error**.

Otherwise, return $(-A1[1]) + R$.

Examples:

In the following examples, **exponent-field-width** is 3.

```

      D ← -1 -0.1 0 0.1 1 °.× 5p0.5
      '||', (9 -3 7 -1 6 3 5 1 3 0 ⌘ D), '||'
| -5.00E-1  -5E-1  -0.500  -0.5  -1 |
| -5.00E-2  -5E-2  -0.050  -0.1  0 |
|  0.00E0   0E0   0.000   0.0  0 |
|  5.00E-2   5E-2   0.050   0.1  0 |
|  5.00E-1   5E-1   0.500   0.5  1 |

```

```

      p 1 0 2 0 4 0 8 0⌘0 4p1
0 15
      31 0 ⌘ 2*100
1267650600228229401496703205376
      31 20 ⌘ .07
      0.070000000000000000666

      4 1 ⌘ -.99 -.89 0 7.5 11.5
-1.0-0.9 0.0 7.511.5

```

Note: Like other axis lengths, the field width $A[1]$ is limited only by index-limit.

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

16 INPUT AND OUTPUT

16.1 INTRODUCTION

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 this standard.

The protocol for the use of a session takes the form of a dialogue: the user makes an **entry** and passes control to the APL system. The system processes the entry, produces a **response**, and returns control to the user.

The user makes entries on a keyboard, and obtains responses by seeing them presented on a display-device. The combination of a keyboard and a display-device is intended to represent, abstractly, a terminal. This standard does not require the use of any particular terminal, keyboard, display or method of encoding **characters**. The ISO 2022.2 APL character encoding reproduced in Appendix A is widely used.

The display-device is considered a window into a **presentation-space**, an unbounded sequence of past entries and responses. The mapping between the **presentation-space** and the display-device is not specified by this standard.

The system obtains entries as **tokens of class constant** or **interrupt** by calling the session operation **read-keyboard**. It presents responses by calling the session operation **display** with a **token of class result**. The transformations from entry to token and from token to response are not specified by this standard, but certain desirable characteristics of these transformations are given as comments.

Because the dialogue protocol described here alternates between user and system, the session is always in one of two logical **keyboard-states**, **open-keyboard** or **locked-keyboard**, depending upon whether the user is making an entry or the system is producing a response. The user can request a change in **keyboard-state**, from **locked-keyboard** to **open-keyboard** by using the **signal-attention** facility. In addition, the user can force a change in **keyboard-state** by signalling **interrupt**.

16.2 DEFINITIONS

16.2.1 User Facilities

A user should have the following facilities.

Edit-Actions: A collection of **implementation-defined** facilities that permit the user to make entries.

Note: Typical **edit-actions** include inserting, deleting, replacing and superimposing graphic symbols within the current entry and combining display-lines from the presentation-space with the current entry.

Enter: A facility, available to the user when the terminal is in **open-keyboard** state, for changing the **keyboard-state** to **locked-keyboard** and returning to the caller of **read-keyboard** either an **interrupt** or a **character vector**. **Enter** sets the value of the session attribute **attention-flag** to **zero**.

Signal-Attention: A facility, available to the user when the terminal is in **locked-keyboard** state, for setting the value of the session attribute **attention-flag** to **one**.

Signal-Interrupt: A facility, available to the user at all times, for interrupting any evaluation sequence, causing it to return an **interrupt token** to the current caller of **evaluate-line**. Any **atomic** operation so interrupted has no effect on the state of the **active-workspace**.

Note: **Signal-interrupt** can be issued in **open-keyboard** state and in **locked-keyboard** state, although the particular means of invoking the operation may be different for the two states. Transmitting the superimposition of the graphics for **O** **U** and **T** is a typical means of signalling **interrupt** in **open-keyboard** state.

16.2.2 Implementation Algorithms

Read-Keyboard: An operation, available to the system when the session is in **locked-keyboard** state, that returns either an **interrupt token** or a **token of class constant** and **content a character vector**.

Note: Read-keyboard should display current-prompt, then change the keyboard-state to open-keyboard to permit the user to use edit-actions to enter a list of characters or to signal an interrupt.

A concept long honoured in APL is that of visual fidelity: the graphic symbols that appear on the display as the result of edit-actions should correspond exactly to those returned to the system as the result of read-keyboard.

Display T : An operation, available when the session is in **locked-keyboard** state, for presenting the **token T** on the **display-device**. T is either a **value** or an **error**.

Note: The following is a suggested, but not required, evaluation sequence for display T :

If **quote-quad-prompt** is not the **empty character vector**,

Append to the **presentation-space** as a new last item a **display-line** whose graphic symbols represent the **characters in quote-quad-prompt**.

Set **quote-quad-prompt** to the **empty character vector**.

If T is an **error**, append to the **presentation-space** as a new last item an indication of

The **class of the error**.

The point in **current-line** at which evaluation was stopped.

If the **mode** is **defined-function**, the **function-name** and **current-line-number** of the **current-context**.

If T is a **value**,

Let A stand for the **content of T** .

If A is **numeric**, display ΦA .

Otherwise,

If A is a **scalar** or **vector**, append to the **presentation-space** as a new last item a **display-line** whose graphic symbols represent the **characters in A** .

Otherwise,

Set I to **zero**.

Set M to $((\times / ^{-1} \uparrow p A), ^{-1} \uparrow p A) p A$.

Repeat the following $1 \uparrow p M$ times:

Set I to $I + 1$.

Display row I of M .

Append $(^{-1} \uparrow p p A) | + / \wedge \backslash 0 = \Phi (^{-1} \uparrow p A) \tau I$ **blank display-lines** to the end of the **presentation-space**.

(End of repeated block)

16.2.3 Prompts

Note: The following implementation-defined arrays are used to indicate to the user the type of entry required.

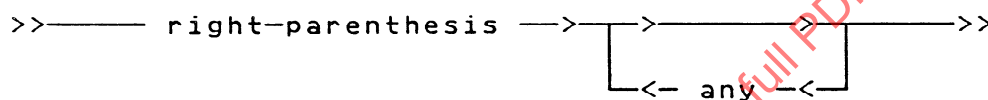
Indent-Prompt: An implementation-defined character array, typically a vector of six blanks.

Quad-Prompt: An implementation-defined character array, whose first row typically begins with '□:' and whose second row is the indent-prompt.

Function-Definition-Prompt: An implementation-defined character array, typically a vector beginning with [nn] where nn is a line-number.

16.3 DIAGRAMS

System-Command-Line



Function-Definition-Line



16.4 OPERATIONS

16.4.1 Immediate-Execution

Immediate-Execution with X .

Informal Description: **Immediate-execution** mode is the primal mode, the first evaluation sequence processed by the system to enter into a dialogue with a user. X is either an **error token** or **nil**.

The system also calls **immediate-execution** when an **error** or the **attention-flag** causes a defined function to be suspended. In this case, **immediate-execution** displays a status indication before prompting for input.

Evaluation Sequence:

Display X .

Repeat:

Set **current-prompt** to **indent-prompt**.

Set E to **read-keyboard**.

If E is a **constant**,

Let Q stand for the **content** of E .

If Q matches **system-command-line**, set T to **evaluate-system-command** Q .

If Q matches **function-definition-line**, set T to **evaluate-function-definition-request** Q .

Otherwise,

Generate a new **context** in which

mode is **immediate-execution**,

current-line is Q ,

current-function is 0, 0p, 1, 1',

current-line-number is **one**,

current-statement is the empty list of tokens, and

stack is the empty list of tokens.

Append the new **context** to the **state-indicator** of the **active-workspace** as a new first item.

Set T to **evaluate-line**.

Remove the **first-item** in the **state-indicator** of the **active-workspace**.

If T is a **branch**, a **clear-state-indicator**, or an **escape** and the **state-indicator** of the **active-workspace** is not an **empty-list**, return T .

If T is an **error** or a **constant**, **display** T .

(End of repeated block)

16.4.2 Quad Input

$$Z \leftarrow \square$$

Informal Description: Z is an array provided by the user in response to a prompt.

Evaluation Sequence:

Repeat:

Set **current-prompt** to **quad-prompt**.

Set E to **read-keyboard**.

If E is an **interrupt**, return E .

Let Q stand for the **content** of E .

If some element of Q is not **blank**,

Generate a new **context** in which

mode is **quad-input**,

current-line is Q ,

current-function is 0 0p ' ',

current-line-number is **one**,

current-statement is the empty list of **tokens**, and

stack is the empty list of **tokens**.

Append the new **context** to the **state-indicator** of the **active-workspace** as a new first item.

Set Z to **evaluate-line**.

Remove the **first-item** in the **state-indicator**.

If Z is **escape**, signal **unwind**.

If Z is **unwind** or **clear-state-indicator**, return Z .

If Z is a **value**, return a **token** with **class** **constant** and **content** that of Z .

If Z is an **error**, **display** Z .

Otherwise, **display value-error**. (see note)

(End of repeated block)

Note: The **display value-error** line in the above evaluation sequence is introduced to permit certain **consistent-extensions**. In the evaluation sequence, errors are **displayed** rather than **signalled**, since signalling would terminate the Repeat loop and force Return.

Quad-input returns only **tokens** of class **escape** and **unwind** (from $\rightarrow$), **clear-state-indicator** (from **)SIC**), **constant** (from $2 * . 5$) and **interrupt**. It reports an error and reprompts for all other **results**. Note that it reprompts without an error report if the input was empty or all blank.

16.4.3 Quote Quad Input

$Z \leftarrow \square$

Informal Description: Z is a **character vector**. Input in response to $\square$ is treated as a **character value**. Z is a vector whose length is the number of positions from the left margin up to the rightmost **character** of the input, including explicitly entered trailing blanks.

Evaluation Sequence:

Set **current-prompt** to **quote-quad-prompt**.
Set **quote-quad-prompt** to the **empty character vector**.
Set E to **read-keyboard**.
Return E .

Note: The behaviour required here differs from that of some existing systems, in which a single character response to $\square$ is returned as a scalar.

16.4.4 Quad Output

$Z \leftarrow \square \leftarrow B$

Informal Description: Z is B . As a side effect, the array B is displayed on the terminal.

Evaluation Sequence:

Display B .
Return a **committed-value** with **content** B .

16.4.5 Quote Quad Output

$$Z \leftarrow \square \leftarrow B$$

Informal Description: Z is B , and **quote-quad-prompt** is set to B .

Evaluation Sequence:

If any **item** of the **ravel-list** of B is not a **character**, signal **domain-error**.

If the **rank** of B is **greater-than one**, signal **rank-error**.

If the **count** of B is **greater-than quote-quad-output-limit**, signal **limit-error**.

If **quote-quad-prompt** is not empty, signal **limit-error**.

Set **quote-quad-prompt** to B .

Return a **committed-value** with **content** B .

Example:

```

      ▽ Z←PROMPT X
[1]   □←X←,␣X
[2]   Z←(ρX)↓□
      ▽
      PROMPT '1 CLEANSPEACE DATE? '
1 CLEANSPEACE DATE? 1966-11-27
1966-11-27

```

Note: Quote-quad output is difficult to standardise because it is implemented in several different ways in existing systems, and, in each system, is heavily used. There is, however, sufficient commonality to permit the inclusion of this restricted form of quote-quad output. Quote-quad output on a **conforming-implementation** should support the prompt function shown in the example.

This page intentionally left blank

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

17 SYSTEM COMMANDS

17.1 INTRODUCTION

Note: Any line of input in **immediate-execution** whose first non-blank character is a right parenthesis is considered to be a system command.

17.2 DEFINITIONS

Global-Referent of T : For T a **classified-name**, the **last-item** in the **referent-list** of the **symbol-named-by T** .

Global-Context: The last **context** in the **state-indicator** of the **active-workspace**.

Library-Workspace-Named A : For **workspace-identifier A** , the **item** of the **library** whose **owner** is **this-owner**, and whose **workspace-name** is A .

Already-Exists: A **workspace** **already-exists** if its **existential-property** is **present**.

Does-Not-Exist: A **workspace** **does-not-exist** if its **existential-property** is **absent**.

Note: The model of libraries used in this standard assumes that a **workspace** object exists in the **library** for all possible **workspace-identifiers**. **Workspaces** that have been dropped or that have never been saved are distinguished from those that have been saved by the setting of their **existential-property**; the **existential-property** of saved **workspaces** is **present**, while that of unsaved **workspaces** is **absent**.

Attempt-to-Erase A : For A , a **simple-identifier**, an operation defined as follows:

If A is **globally-erasable**,

If the **current-class** of A is **shared-variable**, **retract A** .

Set A to **nil**.

Return **command-complete**.

Otherwise, signal **not-erased**.

Copy A from W : For **simple-identifier A** , and **workspace-identifier W** , an operation defined as follows:

Let GA stand for the **global-referent** of A in the **active-workspace**.

Let GB stand for the **global-referent** of A in the **library-workspace-named W** .

Set Z to **attempt-to-erase GA** .

If Z is an **exception**, signal **not-copied**.

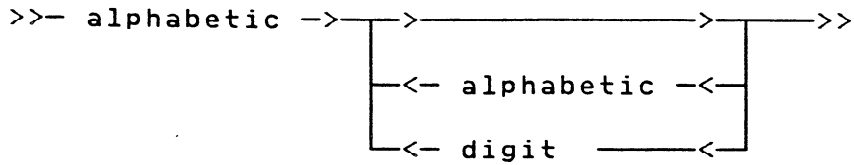
If GB is a **shared-variable**, set GA to the **shared-value** of GB .

Otherwise, set GA to GB .

Return **command-complete**.

17.3 DIAGRAMS

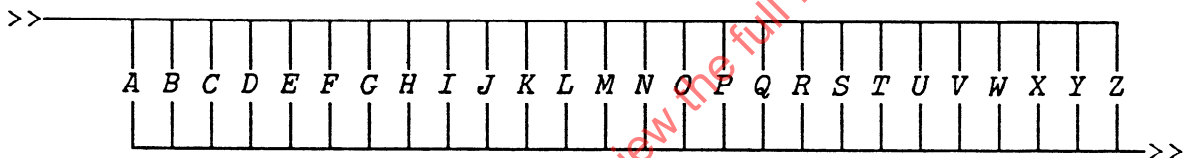
Workspace-Identifier



Example:

CLEANSPACE

Alphabetic



17.4 OPERATIONS

17.4.1 Evaluate-System-Command

Evaluate-System-Command *Q*.

Note: The forms of system commands are not regular enough to permit much generalisation. Therefore, the forms themselves are used as the key to the evaluation sequences.

Evaluation Sequence:

Find an entry that matches *Q* in the **system-command evaluation sequences**.

If no such entry is found, signal **incorrect-command**.

Otherwise, call the corresponding evaluation sequence.

Return the **token** it returns.

STANDARDSISO.COM : Click to view the full PDF of ISO 8485:1989

17.5 DIAGRAMS AND EVALUATION SEQUENCES

In the following,

p stands for **permitted-blanks**,
r stands for **required-blanks**,
w stands for **workspace-identifier**, and
a stands for **simple-identifier**.

Clear Active Workspace

```
>>— ) — p — CLEAR — p —>>
```

Call **Shared-Variable-Reset**.
Set the **active-workspace** to the **clear-workspace**.
Set **comparison-tolerance** to **initial-comparison-tolerance**.
Set **index-origin** to **initial-index-origin**.
Set **latent-expression** to **initial-latent-expression**.
Set **print-precision** to **initial-print-precision**.
Set **quote-quad-prompt** to the **empty character vector**.
Set **random-link** to **initial-random-link**.
Return **command-complete**.

Copy Library Workspace Object

```
>>— ) — p — COPY — r — w — r — a — p —>>
```

If the **library-workspace-named w** **does-not-exist**, signal **not-found**.
If the **global-referent** of a in w is not a **defined-function**, **niladic-defined-function**, **variable**, or **shared-variable**, signal **incorrect-command**.
Set Z to **copy a from w**.
Return Z.

Copy Library Workspace

```
>>— ) — p — COPY — r — w — p —>>
```

If the **library-workspace-named w** **does-not-exist**, signal **not-found**.
For each **symbol S** of the **library-workspace-named w**
 Let A stand for the **name** of S.
 If the **global-referent** of A in w is a **defined-function**, **niladic-defined-function**, **variable**, or **shared-variable**,
 Set Z to **copy A from w**.
 If Z is an **exception**, return Z.
Return **command-complete**.